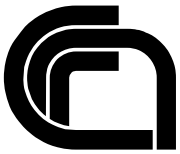

Consiglio Nazionale delle Ricerche

A Fast and Accurate Heuristic for the Single Individual SNP Haplotyping Problem with Many Gaps, High Reading Error Rate and Low Coverage

L. M. Genovese, F. Geraci, M. Pellegrini

IIT TR-05/2007

Technical report

Maggio 2007

Istituto di Informatica e Telematica

A Fast and Accurate Heuristic for the Single Individual SNP Haplotyping Problem with Many Gaps, High Reading Error Rate and Low Coverage

Loredana M. Genovese, Filippo Geraci and Marco Pellegrini

Istituto di Informatica e Telematica del CNR, Via G. Moruzzi 1, 56100-Pisa (Italy).
loredana.genovese@gmail.com, filippo.geraci@iit.cnr.it, marco.pellegrini@iit.cnr.it

Abstract. Single nucleotide polymorphism (SNP) is the most frequent form of DNA variation. The set of SNPs present in a chromosome (called the *haplotype*) is of interest in a wide area of applications in molecular biology and biomedicine, including diagnostic and medical therapy. In this paper we propose a new heuristic method for the problem of haplotype reconstruction for (portions of) a pair of homologous human chromosomes from a single individual (SIH). The problem is well known in literature and exact algorithms have been proposed for the case when no (or few) gaps are allowed in the input fragments. These algorithms, though exact and of polynomial complexity, are slow in practice. Therefore fast heuristics have been proposed. In this paper we describe a new heuristic method that is able to tackle the case of many gapped fragments and retains its effectiveness even when the input fragments have high rate of reading errors (up to 20%) and low coverage (as low as 3). We test our method on real data from the *HapMap Project*.

1 Introduction

The single nucleotide polymorphism or SNP (pronounced "snip") is the most common variation in the human DNA. In fact a recent study of 2001, has shown that similarity among human DNA sequences is over 99% and only a few bases (just 1.42M bases overall) are responsible for the variations in human phenotypes [13].
A SNP is a variation of a single nucleotide in a fixed point of the DNA sequence and in a bounded range of possible values. The sequence of SNPs in a specific chromosome (or a large portion of a chromosome) is called generically *Haplotype*. Since most cells in humans are *diploid*, each chromosome (except the X and Y chromosomes in males) comes in two almost identical copies, one inherited from the mother and one from the father. Thus the haplotype of a chromosome is fully described by two sequences of SNPs in the two copies of the chromosome.
The Single Individual SNP Haplotype reconstruction problem may be viewed as the problem of rebuild the two strings forming the haplotype from a set of fragments obtained by shotgun sequencing of the chromosomes' DNA strands. The most important aspect of the problem is that with the current technology it is difficult and/or impractical to keep trace of the association of the fragments with their chromosome, thus this association has to be reconstructed computationally and it is a preliminary necessary phase to the actual fragment assembly to reconstruct the haplotype.
Unlike the classical DNA fragment assembly problem, in which the position and orientation of fragments is unknown, in the parental haplotype reconstruction problem the position of each fragment is fixed and known. Further aspects that must be considered that render the problem difficult (and computationally interesting) are the following:

1) **Reading errors**. The complex nature of the biological/chemical/optical processes involved in shotgun sequencing implies that a non negligible error probability is attached to each single SNP reading.

2) **Coverage of fragments**. Algorithms using fragments to reconstruct a string rely heavily on the fragment's overlaps and on the redundancy of information provided by several fragments covering the same SNP position, to perform in silico correction of reading errors. Thus a critical parameter of the input data is the minimum (or average) coverage of SNPs by fragments. This number is also related to the throughput of the sequencing equipment.
3) **Gaps in fragments**. Ideally each fragment covers consecutive SNP positions in the order of the SNPs of a chromosome. However in practice we may have many fragments with gaps due to several phenomena.
   3.1 AMBIGUOUS READINGS. In the reading of fragments it may happen that is impossible to detect the value of a SNP with a sufficient confidence. It is better to model this ambiguous case with a small gap rather than introduce spurious values.
   3.2 MATEPAIR SEQUENCES. Some shotgun sequencing methodologies produce pairs of fragments that are from the same chromosome, do not overlap and whose distance is known up to a certain degree of precision. Matepair sequences are used to cope with the presence of repeat subsequences that complicate the reconstruction efforts This extra information attached to the produced fragments can be considered logically equivalent to a single fragment with one gap.

Our contribution In this paper we propose a heuristic algorithm for the SIH problem that is fast, handles well gaps, and is able to deal with high reading error rates and low fragment coverage. We demonstrate these properties via experiments on real human data from the *HapMap project* [6]. Advanced Personalized Medicine is one of the goals of current research trends and in this area new genetic diagnostic methods are critical. It is thus important to support diagnostic technologies that can be used as much as possible in the field (closer to the patient, and far from the traditional high tech labs). Away from the controlled environment of a lab it is likely that the current portable technology for sequencing will produce less reliable data. Moreover, if a real time and high throughput response is needed to care for the needs of many individuals in a short time span, one might not be able to guarantee a high coverage of the fragments and low reading error rates.. Our algorithm is a step forward in the direction of extracting efficiently useful information even from low quality data.

1.1 Formalization of the problem

From the computational point of view the problem of the haplotype reconstruction was defined in [8, 14]. It can be easily described as follow: let $S = s_1, s_2, \ldots, s_n$ a set of SNPs (specific positions in a DNA string) and $F = f_1, f_2, \ldots, f_m$ a set of DNA fragments. Each SNP can be covered by a certain number of fragments and can take only two values (The values of the haplotype in that position). The natural way of representing fragments is to store them in an m x n matrix M called *SNP matrix*. The element $M_{i,j}$ contains the value of the SNP s_j in the fragment f_i or the special character $-$ if that SNP is unspecified in the fragment. If the element $M_{i,j} = -$ we say that it is a *hole* or, equivalently, that the fragment f_i contains an hole at position j. Let $f_i \in F$ and $1 \leq a \leq b \leq n$ such that $\forall k \in [a, b]$, $M_{i,k} \neq -$ and $\forall k \in [1, m]/[a, b]$, $M_{i,k} = -$, the fragment f_i is called *gapless*. We say that M is gapless if all its fragments are gapless.
We say that two fragments f_i and f_j have a *collision* if the following condition is true: $\exists k \in [1, n]$ such that $M_{i,k} \neq M_{j,k} \wedge M_{i,k} \neq - \wedge M_{j,k} \neq -$. Given the matrix M the conflict graph $G = (V, E)$ is defined as follow: for each row of M there is a vertex labelled with the correspondent fragment f_i. If f_i has no collision with f_j, insert an edge between V_i and V_j. An example of conflict graph is in Figure 1. When the matrix M does not contains errors G is bipartite. In this case the haplotype reconstruction is easy to solve. The rows of M can be split in two disjoint sets according to the

H_1 = ATTACCTT
H_2 = GATCGGAT

f_1 = A-TA--TT
f_2 = -TTGC-A-
f_3 = A--A-C-T
f_4 = GA-CG-AT
f_5 = -----GA-
f_6 = -AT-AG--

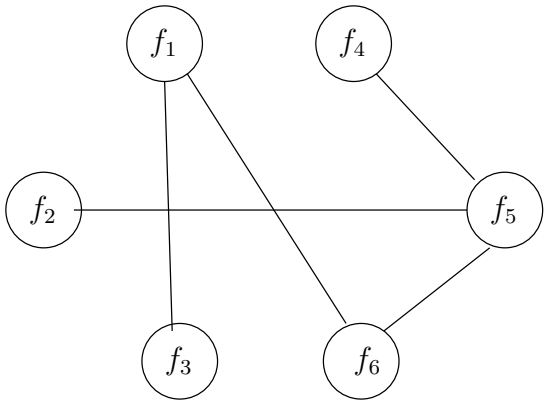

Fig. 1. On top left a couple of haplotypes, on bottom left the matrix M with the fragments (containing errors and gaps) and on the right the corresponding conflict graph.

bipartition of G. By construction of graph G the i-th character of all elements of a set induced from bipartition have the same value or is a gap. Thus for each set we build an haplotype simply choosing as value for SNP s_i the value of the i-th character (not equal to $-$). If M is not error free, the graph G may be not bipartite. The single individual haplotype reconstruction problem, can be reduced to one of the following problems [2]:

- Minimum Fragment Removal (MFR): determine a minimal number of fragments (rows of the matrix M) whose removal from the input set induces a bipartite graph.
- Minimum SNP Removal (MSR): determine a minimal number of SNPs (columns of the matrix M) whose removal from the input set induces a bipartite graph.
- Longest Haplotype Reconstruction (LHR): determine set of fragments (rows of the Matrix M) whose removal from the input set induces a bipartite graph and the length of the induced haplotype is maximized.
- Minimum Error Correction (MEC): determine a minimal set of entries of the matrix M whose correction to a different value induces a bipartite graph.

Our approach. We give a heuristic method for the minimum error correction problem MEC. It is a heuristic method since we have no guarantee of attaining the minimum, nor any guarantee on the approximation to the minimum that we can achieve. Note however than MEC is the hardest of the problems listed above. Our method is organized in phases (four phases) and it is greedy in nature (making choices that are optimal in a local sense). In each phase we perform three tasks: 1) detect likely positions of errors 2) allocate fragments to the two partially built haplotype strings, and 3) build partial haplotype string deciding via majority on ambiguous SNPs. The difference among the three phases is twofold: on one hand we can *use the knowledge built up in the previous phases*, and on the other hand in passing from one phase to the next we *relax* the conditions for the decisions to be taken regarding tasks 1), 2) and 3).

1.2 Organization of the paper

In Section 2 we review the state of the art for the SIH problem. In Section 3 we describe our algorithm. In Section 4 we describe the experiments and their results.

2 State of the art

SNP's and Haplotypes have become recently a focus of research (See the *HapMap project* [6]) because of their potential for associating observable phenotypes (e.g. resilience to diseases, reactivity to drugs) to individual genetic profiles [16]. The technology for detecting the position of SNP's in the human genome has been developed [13, 10] and continues to be refined to produce more accurate SNP maps. Two large and active areas of research involving haplotypes are the determination of the genetic variability in a population (see surveys in [2],[7]) starting from *genotyping* data, and the association of genetic variability with phenotypes.

In this paper we discuss the problem of determining the haplotype of a single individual based on fragments from shotgun sequencing of his/her DNA which is known as the *Single Individual SNP Haplotyping Problem* (SIH)[1]. This problem has been tackled both from a theoretical point of view [8, 14, 3, 1, 4] and from a more practical one [15, 12, 9]. Weighted versions of the problem are studied in [17].

The SIH problem is clearly not formally an input/output problem as defined usually in computer science[2], therefore precise complexity statements can be made only for the derived problems such as: MEC, LHR, MFR and MSR. MEC even with gapless fragments is NP-hard [3], and it is APX-hard for fragments with at most 1 gap [4]. There is an $O(\log n)$-approximate polynomial time algorithm [12]. LHR with gapless fragments can be solved exactly in polynomial time [3]; it is NP-hard and APX-hard for fragments with at most 1 gap [4]. MFR is NP-hard for fragments with at most 1 gap, and MSR is NP-hard for fragments with at most 2 gaps [8]. If we have a bound k on the total number of gaps, for k constant, MFR and MSR are polynomially solvable [14]. In general MFR and MSR are APX-hard.

The polynomial time algorithms proposed for the above problems are at least cubic (in the gapless case) therefore a faster heuristic method has been proposed in [12] that is based on an incremental construction. We improve upon [12] by giving a method that is as fast in practice and more accurate when the reading error rate increases and/or the fragment coverage decreases. Interestingly, even if exact polynomial algorithms are known for MFR on gapless input in [14], simulation data show that the heuristic method of [12] achieves better accuracy in solving the original SIH problem. For this reason we take [12] as baseline algorithm even when dealing with fragments with gaps.

Wang et al. [15] describe a Genetic Algorithm for this problem that in some reported experiments gives good performance for short haplotypes (about 100 SNPs). It is unclear how this method would performs on longer haplotypes and with lower coverage rate. As future work we plan a comparison of our method with the one in [15].

3 Our Heuristic

We start by building the SNP matrix M with m rows and n columns where each row is a fragment. The element in position $M_{i,j}$ is the j-th SNP in fragment f_i or $-$, if it is a gap. Our heuristic builds the haplotype consensus with a pre-processing (phase zero) and three main phases (1-3):

Ph-0 We perform a statistical analysis of potential conflicts among pairs of columns in M;

Ph-1 in phase *Ph-1* we select a first group of columns with the highest possible confidence to be error-free and we build an initial solution from them;

[1] Also called the Haplotype Assembly Problem.

[2] SIH informally relates the output of the algorithm to an unknown DNA string whose "approximation" is the purpose of the algorithm. The formal input to the algorithm is a set of fragments that are related to the unknown string via physical error-prone processes. Thus there is no mathematically formalized relationship between the input and the criterion for evaluating the output of the algorithm.

Ph-2 in the second phase we select those columns that we are able to disambiguate using the solution obtained in the previous phase;

Ph-3 in the last phase we try to complete the solution using weaker conditions for assigning columns to the final solution.

In this section we will give priority to an intuitive understanding of the several phases and steps, skipping on some more formal details to be expanded in the full paper.

3.1 Phase zero: Preprocessing

For each column of M we build a *group* G_i containing a certain number of *sets*. Each set is initialized with the indexes of all the rows which have in position i a character different from $-$. So G_i can contain from 0 up to 4 sets (the empty set and one for each base: a, c, g, t).

Observation 1 *If G_i has 0 sets, column i is empty. In this case there is no data to reconstruct the haplotype for column i. If G_i has just 1 set, all the character in column i are the same. If G_i has more then 2 sets, column i contains errors.*

If G_i contains three of four sets, we can suppose that the one or two smaller sets are due to errors. Unfortunately we can only detect the presence of errors, but we have not enough information to correct them. In this case we remove from the matrix M the information about the possibly uncorrect values and update G_i accordingly. Note that in cases where G_i contains a large set and two smaller ones of the same size, we can not remove those sets because we could likely be removing correct data.

If we suppose a constant coverage of each locus by both the haplotypes, in the case G_i has two sets and one of them is much bigger than the other, we can suppose that locus to be homozygote and the data in the smaller set is a reading error. Clearly in this case we can predict the right content of the matrix M in these positions.

After filtering out the above easy cases we are left to deal with groups of two sets of non negligible size. Given two groups $G_i = (S_{i,1}, S_{i,2})$ and $G_j = (S_{j,1}, S_{j,2})$ having exactly 2 sets and such that $i \neq j$, we call *conflict matrix* the squared matrix $E_{i,j}$ of order 2:

$$E_{i,j} = \begin{pmatrix} S_{i,1} \cap S_{j,1} & S_{i,1} \cap S_{j,2} \\ S_{i,2} \cap S_{j,1} & S_{i,2} \cap S_{j,2} \end{pmatrix}$$

When only one diagonal of E has its elements non-zero and it is of full rank, there are no detectable errors. Otherwise we have a conflict between column i and j. The detected errors could be in one or both columns.

Observation 2 *If $E_{i,j}$ has only one element equal to $\emptyset$ we can suppose that the corresponding diagonal element contains the reading errors and its cardinality is the number of such errors. For example if in $E_{i,j}$ only the element $S_{i,2} \cap S_{j,1}$ is 0 then there are $|S_{i,1} \cap S_{j,2}|$ errors in at least one of the columns i and j in the rows of indexes in $S_{i,1} \cap S_{j,2}$. The assumption that the elements in $S_{i,1} \cap S_{j,2}$ are the errors in $E_{i,j}$ becomes more plausible if its cardinality is significantly smaller than the others.*

Observation 3 *In presence of errors in $E_{i,j}$ we can not establish if the error is in column i or j or both. We can locate the error if one of the following conditions hold:*

- *If $\forall k \neq j$ $E_{\min(i,k),\max(i,k)}$ does not contain errors, than errors must to be in the column j*

– *If $\exists k$ such that $E_{i,j}$ has an error, $E_{\min(j,k),\max(j,k)}$ has an error and $E_{\min(i,k),\max(i,k)}$ has no errors, than we deduce that the error is in the j-th column.*

In the case of the example in observation 2, if also one of the conditions of observation 3 holds, we deduce that the errors are in the rows $S_{i,1} \cap S_{j,2}$ in the column j. So we can correct the error by removing from M the incorrect values and updating $S_{j,2}$ via removing $S_{i,1} \cap S_{j,2}$.

If none of the conditions in observation 3 hold we can not discriminate between columns i and j so we can remove the errors at the cost of a loss of information by assigning: $S_{j,2} = S_{j,2} \setminus S_{i,1} \cap S_{j,2}$ and $S_{i,1} = S_{i,1} \setminus S_{i,1} \cap S_{j,2}$.

We have observed empirically that the error correcting criteria of Phase 0 are effective when the input has a very low reading error rate. As the error rate increases the bulk of the disambiguation is on the next phases 1-3.

3.2 First phase

The main goal of the first phase is the selection of a set of pair of groups with the highest possible probability of containing no inconsistencies and extract from them two sets of fragments that will be the core of the first (partial) solution.

Candidate list selection The optimal set of candidate pairs to select is that in which each group has no conflicts with all the other groups. Unfortunately, if the percentage of errors in M is high this set can be empty. Moreover a correct group can be involved in a conflict with another group due to reading errors in the latter. This cause the removal of all the pairs in which that group appears. Higher coverage tends to increase this bad effect on the size of the candidate set. In fact the probability that a group with no errors has a conflict with a group with errors is proportional to the coverage.

If the optimal candidate set of groups pairs is empty, we must to find the set with the highest confidence to be a good candidate set. First of all we compute the mean number of conflicts among pairs of groups. As candidate set we pick all the pairs for which all the following conditions hold:

a) both its groups have two sets,
b) the number of conflicts in which its groups are involved is less than the mean,
c) the matrix E of its groups is diagonal and of full rank

Extraction of initial core From the candidate list obtained in the previous paragraph, we build now two disjoint set of rows of M that will be used as core of the final solution.

We build a series of chains of pair in this way: the first pair of a new chain is the first unused pair of the candidate list. Then we add a pair to the chain if at least one group of the pair is already in the chain until no more pairs can be added to the chain. The procedure stops when all the pairs are in a chain. At the end we select the longest chain.

The construction of the series of chains is straightforward. First of all, we sort the candidate pairs in lexicographic order and place them in a vector $L = [C_0, \ldots, C_{|L|}]$. We build also a vector V in which we store all the indexes $j \in [1, |L|]$ of L such that the the first elements of consecutive pairs are different, $C_j[0] \neq C_{j-1}[0]$. We set as first element of V the value 0 and as last element of V the value $|L|$.

We build also a vector v of size m containing several status flags: position i is set to "to visit" if the group i-th does not appear in any chain, set to "visited" if it appears in the chain we are building and set to "complete" if all the pair containing the index i were already used. A new chain is built as follows:

1. Find an index i such that the pair C_i is not already used and set it as the first element of the chain.
2. All the elements of L not yet used in the range $[V[i], V[i+1]-1]$ are added to the chain, if they exist. The vector v is updated accordingly.
3. If there is an index j such that $v[C_j[0]]$ is set to "visited" got step (2.) using i such that $V[i] = j$. Otherwise search a pair where $v[C_j[1]]$ is set to "visited" and goto step (2.) using i such that $V[i] = j$.

1. if v has no element set to "visited" the chain is complete.

It is easy to note that the arbitrary choice of the first element do not influence the pairs that will fall in the chain, but only their order that is not important in our heuristic.
Chains have the important propriety:

Property 1 *If we consider groups in the same order in which they appear in a chain, one of the following conditions holds:*

1. $S_{i,1} \cap S_{i+1,1} \neq \emptyset \wedge S_{i,2} \cap S_{i+1,2} \neq \emptyset \wedge S_{i,1} \cap S_{i+1,2} = \emptyset \wedge S_{i,2} \cap S_{i+1,1} = \emptyset$
2. $S_{i,1} \cap S_{i+1,2} \neq \emptyset \wedge S_{i,2} \cap S_{i+1,1} \neq \emptyset \wedge S_{i,1} \cap S_{i+1,1} = \emptyset \wedge S_{i,2} \cap S_{i+1,2} = \emptyset$

We are now ready to build a sort of "super-group" $\mathcal{G} = (\mathcal{S}_1, \mathcal{S}_2)$ in which $\mathcal{S}_1$ will be used to build the first haplotype consensus, and $\mathcal{S}_2$ for the second haplotype. If G_0 is the first element of the longest chain, $\mathcal{S}_1$ is initialized with the elements of $S_{0,1}$ and $\mathcal{S}_2$ is initialized with the elements of $S_{0,2}$.

Property 1 suggests a simple way to assign the sets of each considered group to a set $\mathcal{S}_i$. In fact if, for example, the elements in set $S_{i,1}$ are assigned to $\mathcal{S}_1$ and $S_{i,1} \cap S_{i+1,1} \neq \emptyset$ holds, the elements in set $S_{i+1,1}$ can also be assigned to $\mathcal{S}_1$.

All the groups whose sets are assigned to $\mathcal{G}$ are marked as "used" and will not be considered in the next phases.

If the considered columns of M, (remember that G_i refers to column i of M), have no errors we have that $\mathcal{S}_1 \cap \mathcal{S}_2 = \emptyset$. Otherwise[3] there are errors in the rows of M whose indexes are in the intersection and in at least one of the columns considered. If there is an element j in both the $\mathcal{S}_i$'s and we do not remove it from one of these sets, the fragment f_j would give its contribute to both the haplotypes, which is incorrect. In order to choose to which haplotype to assign the fragment f_j, we simply count how many times j appear in the sets assigned to $\mathcal{S}_1$ and $\mathcal{S}_2$ and assign j to the set with the highest number of assignments.

3.3 Second phase

If we succeed in partitioning all the rows of M we are ready to build the final haplotype consensus using the method in sub-section 3.5. Experiments with high error rates show that at the end of the previous phase we are able to assign a large part of the rows of M, but not all of them because we had not enough information to unambiguously assign some fragment to a set $\mathcal{S}_i$.

In this phase we already have a partial solution that could give us more information and we can use weaker conditions to assign elements of the groups to $\mathcal{G}$. The first information we distill from the partial solution is an estimation of the mean ratio between the cardinality of sets of the fragments belonging to the two haplotype strings. We compute this ratio only for those groups that were involved in the partial solution because they have higher probability to be correct with respect to the others. We can now safely assume that if the ratio between the cardinality of the sets of an unused group is far enough from the mean, the locus represented from that group is

[3] As before, high reading error rates reduce the efficacy of previous filtering steps.

homozygote and the elements in the smaller set of that group are all errors and can be corrected updating M accordingly.

Considering $\mathcal{G}$ as a group, we can build a vector of conflict matrices $E = E_1, \ldots, E_n$, such that E_i is the conflict matrix relative to $\mathcal{G}$ and G_i. Note that these matrices are more informative than those of previous phase because they are representative of a greater part of the input and not only of two columns. In case of conflicts in E_i we can with high probability say that the errors are in G_i and not in $\mathcal{G}$. This becames more evident in the case of a matrix E_i that have just one element equal to 0 and the value in the diagonal with the 0 is much smaller than the values in the other diagonal. A matrix of this form was discarded in the previous phase, because the error position was not predictable with enough confidence. Instead, here the information provided by $\mathcal{G}$ gives us the ability to deduce the exact position of the errors in the i-th column of M and correct them.

The main goal on this phase is to add as many possible elements to G_i trying to correct some errors in M for improving the haplotype consensus. The procedure acts as follow:

1. Let $\alpha = \emptyset$, $\beta = \emptyset$
2. For all those groups G_i with $i \in [1, n]$ not yet marked, with 2 sets and such that E_i is diagonal and of full rank: if $S_{i,1} \cap \mathcal{S}_1 \neq \emptyset$ *and* $S_{i,1} \cap \mathcal{S}_2 = \emptyset$ add the elements of $S_{i,1}$ to α and $S_{i,2}$ to β. Otherwise, due to the fact E_i is diagonal, must hold that $S_{i,2} \cap \mathcal{S}_1 \neq \emptyset$ *and* $S_{i,2} \cap \mathcal{S}_2 = \emptyset$. In this case simply add the elements of $S_{i,1}$ to β and $S_{i,2}$ to α. G_i becomes marked.
3. If an element j appear in both α and β, we simply count how many times j is present in the sets assigned to α and β and assign j to the set with the highest number of assignments.
4. Assign all the elements of α to $\mathcal{S}_1$ and the elements of β to $\mathcal{S}_2$.
5. Recompute the conflict matrix for the groups that are still not marked and restart from step (1.) until no more groups can be marked.
6. Correct errors that can be detected in M and restart from step (1.) until no more groups can be marked.

3.4 Third phase

At the end of phase two, if there is some other group that is not marked yet, there is no further weaker condition that we can use to add those groups to $\mathcal{G}$ safely. The goal in the third phase is not to add elements to $\mathcal{G}$ one by one, but to build another super-group $\mathfrak{G}$ from the remaining unmarked groups and merge it with $\mathcal{G}$, if possible. This strategy relies on the fact that an aggregation of columns is more robust to errors with respect to a single column. The choice to reuse the previous phases seems the most reasonable, but we must use weaker constraints.

We can not use the techniques of the first phase to initialize $\mathfrak{G}$ because at the end it could not intersect $\mathcal{G}$ (or the intersection could be too small). The problem of the intersection between $\mathcal{G}$ and the $\mathfrak{G}$ is important. In fact if all the sets of both have null intersection there is not a way to join $\mathcal{G}$ and $\mathfrak{G}$. Instead, if the intersection is small, because of errors, by mistake we can join each set of $\mathcal{G}$ with the wrong one of $\mathfrak{G}$.

The safest way to initialize $\mathfrak{G}$, is selecting the unmarked group with the highest possible intersection with $\mathcal{G}$. Analyzing the matrices E_i from the previous phase for the unmarked groups, the one with the highest sum of the elements in a diagonal is the best candidate to initialize $\mathfrak{G}$.

After the initialization of $\mathfrak{G}$, we can use the previous phase to add other elements. Here two constraints are relaxed: it is no more necessary that the conflict matrices are of full rank; detected errors in M are not corrected, but simply the wrong data is removed from M.

Let a and b such that $|\mathfrak{S}_a \cap \mathcal{S}_1| > |\mathfrak{S}_a \cap \mathcal{S}_2|$ and $|\mathfrak{S}_b \cap \mathcal{S}_1| < |\mathfrak{S}_b \cap \mathcal{S}_2|$ and $a \neq b$, we assign to $\mathcal{S}_1$ all the elements of $\mathfrak{S}_a$ not in $\mathcal{S}_2$ and assign to $\mathcal{S}_2$ all the elements of $\mathfrak{S}_b$ not in $\mathcal{S}_1$.

3.5 Haplotype consensus

At the end of the previous phase, some fragments could still be assignable to both haplotype strings. They will be assigned a posteriori after the process of consensus construction to the most similar haplotype. We split M in two sub-matrices: M_1 containing all the rows with indexes in $\mathcal{S}_1$ and M_2 containing the rows with indexes in $\mathcal{S}_2$ Naturally it is impossible to establish which of the parent's haplotype is deduced from $\mathcal{S}_1$ and which from $\mathcal{S}_2$.

We call *pivot* of M at position i the element Pv_i^M (different from a hole) that appears more frequently. If the column i of M has no elements, its pivot will be a hole.

The consensus haplotype induced from $\mathcal{S}_1$ is a sequence in which the i-th element is $Pv_i^{M_1}$ and the consensus haplotype induced from $\mathcal{S}_2$ is obtained in the same way from M_2.

4 Experiments

In our experiment we compared the following algorithms:

A) Our heuristic, as described in section 3;
B) Our implementation of Fast Hare following the description in [12];
C) The trivial reconstruction algorithm by majority voting that has the true fragment assignment as part if its input (*Baseline*).

We implemented the algorithms in Python. Tests have been run on a Intel(R) Pentium(R) D CPU 3.20GHz with 4GB of RAM and with operating System Linux. All algorithms completed their task in less than 10 seconds for the data of largest size considered (strings of 1000 SNP's).

4.1 Input data and fragment generation

In previous papers [8, 12] experiments were based on SNP matrices obtained from the fragmentation of artificially generated haplotype data. The most common approach to the generation of the SNP matrices was suggested in [11]. The recent research project *HapMap* [13] has produced a map of the human haplotypes that is now publicly available [5]. Thus we were able to generate the fragment matrices from real data instead of using synthetic input haplotypes. Using real data, the Hamming distance between the two haplotypes is not a free parameter of our choice in the generation of M. For the extraction of the SNP matrix from the haplotypes we were inspired by the approach suggested in [11] taking in account standard parameters in current technology for shotgun sequencing. The free parameters we set in our experiments are: (a) the length l of the haplotype section to be reconstructed, (b) the coverage c of each haplotype and (c) the error rate e. Current technology for shotgun sequencing is able to manage fragments of the order of one hundreds of bases. In Li et al.[9] the average distance in bp of two SNP's in the DNA sequence is quantified as 300 bp on average, and each fragment is of 650 bp's. Each fragment covers a number of SNP's in the range roughly $[3, 7]$, thus we chose the length of each fragment in this range.
Our generation schema is as follow for each experiment: we select the haplotype strings from a random chromosome among the human chromosomes numbered in [1..22] (thus excluding the gender chromosomes), we get a contiguous substring of length l from the first haplotype starting from a random location and its homologous substring from the second haplotype. As in [11] each such string is replicated c times. Next, errors are inserted uniformly at random in the haplotype substrings with probability e. At this point the strings are split in fragments by selecting iteratively the next cut point at an integer distance from the previous one chosen uniformly at random in the range $[3, 7]$, starting from the first base. Note that the number of fragments is not determined a

priori but it depends on the length l, on the coverage c and on the distribution of the fragment lengths.
Gaps came from two sources. **Input SNP gaps** are those present in the original *HapMap* data. **Mate pairs** are obtained as follows: random pairs of disjoint fragments belonging to the same haplotype string are mated in a single gapped fragment (at the end of this phase globally 50% of the fragments are 1-gapped).

4.2 Outcome of the experiments

We investigate the performance of our algorithm in different settings varying the input parameters. We choose three different length for the haplotyes: 100 bases as in [12], 500 bases like in [11] and 1000 bases. To test the effectiveness of the method we vary the coverage of each haplotype from 3 to 10 considering that in most reported experiments the coverage is about 5 [11]. To test algorithms robustness we used different levels of errors: from 0% to 20%. Each test was repeated 100 times and in table 1 is reported the mean number of errors in the reconstructed haplotypes with respect to the strings before error implants.

Errors %	Algorithm	Coverage. $l = 100$				Coverage. $l = 500$				Coverage. $l = 1000$			
		3	5	8	10	3	5	8	10	3	5	8	10
0%	Baseline	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Our	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Fast Hare	0.00	0.00	0.00	0.04	0.72	0.40	0.04	0.27	8.94	1.79	2.24	0.04
5%	Baseline	0.97	0.10	0.00	0.00	4.81	0.47	0.01	0.00	11.60	0.99	0.04	0.01
	Our	0.97	0.15	0.07	0.02	5.60	0.57	0.01	0.04	14.95	1.59	0.13	0.03
	Fast Hare	1.26	0.18	0.19	0.03	11.79	0.95	0.03	0.03	21.24	2.97	0.26	0.55
10%	Baseline	4.05	0.75	0.03	0.01	21.14	3.74	0.24	0.03	43.38	7.85	0.60	0.13
	Our	5.39	0.88	0.44	0.03	26.45	4.28	0.33	0.07	60.87	9.95	2.59	0.29
	Fast Hare	9.32	1.54	0.41	0.02	45.52	5.91	0.43	0.07	123.92	15.17	1.43	0.46
15%	Baseline	9.78	2.26	0.34	0.08	47.71	12.57	1.49	0.38	95.40	25.42	2.66	0.77
	Our	12.21	2.83	0.43	0.25	66.55	25.70	2.21	0.96	134.74	35.61	4.59	2.46
	Fast Hare	18.41	3.40	1.55	0.83	102.60	25.34	2.65	0.88	268.09	58.81	4.63	1.59
20%	Baseline	15.13	5.71	1.19	0.35	80.97	27.90	5.04	1.68	159.74	56.90	10.86	3.53
	Our	20.44	7.77	2.16	0.93	120.53	52.38	10.17	4.74	220.52	94.26	23.12	13.54
	Fast Hare	32.63	11.51	3.40	1.68	224.46	64.14	12.32	4.16	469.54	150.18	22.21	11.05

Table 1. Quality measurements on the compared algorithms. Mean over 100 runs of the number of errors in the reconstructed haplotypes for error rate in [0.0,0.2], coverage in [3,10], and haplotypes length $l = 100, 500, 1000$.

Analysis of the experiments. In absence if errors (but with gaps) our method was able to reconstruct the haplotypes exactly in all cases. The reconstruction error rate increases for all three methods as the reading error rate increases and it decreases with the increase of coverage. In order to give a synthetic view of the data in Table 1 we use the *Merit Function* f:

$$f = \begin{cases} 0 & \text{if } Our = FH \\ 1 - \frac{Our-B}{FH-B} & \text{if } Our < FH \\ -\left(1 - \frac{FH-B}{Our-B}\right) & \text{if } Our > FH \end{cases} \quad (1)$$

where Our is the error count of our algorithm, FH is the error count for Fast Hare and B is the error count for the baseline algorithm. Note that when Our and FH tie f has value zero.

When *Our* is better than FH, f assumes a value in the range $[0, 1]$, the higher the absolute value, the better is our algorithm w.r.t. Fast Hare. Symmetrically when Fast Hare is better than Our algorithm f assumes values in the range $[-1, 0]$ the higher the absolute value, the better is Fast Hare w.r.t. our algorithm. This indicator is almost always in our favor (see Figure 2). With high coverage (8,10) and high error, Fast Hare and our method substantially tie on very long strings since the difference in absolute reconstruction error is about one/two SNPs over 1000.

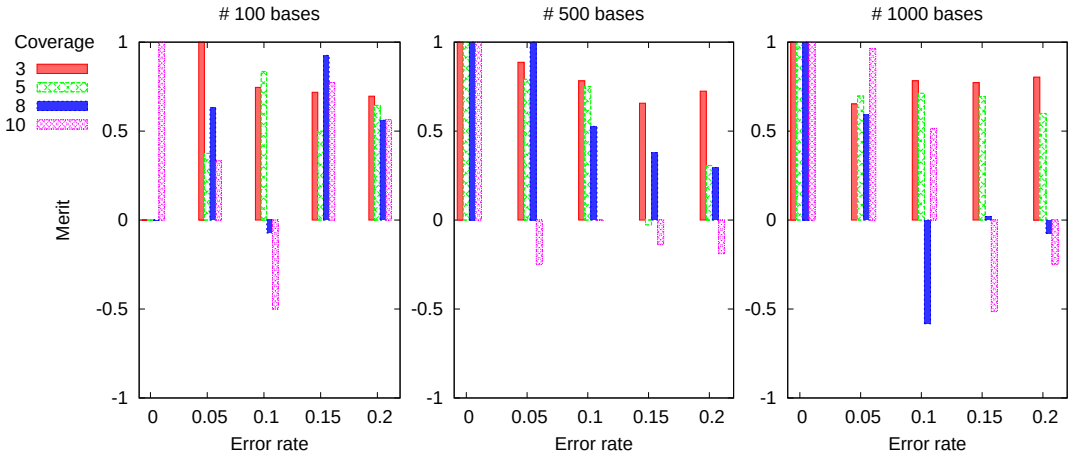

Fig. 2. Figure of merit (see equation 1) for the experiments in Table 1. Values above 0, indicates better relative performance of Our Method over Fast Hare. Values below 0 indicate better relative performance of Fast Hare.

References

1. Vineet Bafna, Sorin Istrail, Giuseppe Lancia, and Romeo Rizzi. Polynomial and APX-hard cases of the individual haplotyping problem. *Theor. Comput. Sci.*, 335(1):109–125, 2005.
2. Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Jing Li. The haplotyping problem: an overview of computational models and solutions. *J. Comput. Sci. Technol.*, 18(6):675–688, 2003.
3. Rudi Cilibrasi, Leo van Iersel, Steven Kelk, and John Tromp. On the complexity of several haplotyping problems. In *WABI*, pages 128–139, 2005.
4. Rudi Cilibrasi, Leo van Iersel, Steven Kelk, and John Tromp. On the complexity of the single individual SNP haplotyping problem. *Algorithmica*, 2007. In print.
5. The International HapMap Consortium. http://snp.cshl.org.
6. The International HapMap Consortium. A haplotype map of the human genome. *Nature*, 437:1299–1320, 2005.
7. D. Gusfield and S.H. Orzack. *Haplotype Inference*, chapter 1, pages 1–25. In CRC Handbook on Bioinformatics. CRC Press, 2005.
8. Giuseppe Lancia, Vineet Bafna, Sorin Istrail, Ross Lippert, and Russell Schwartz. SNPs problems, complexity, and algorithms. In *Proceedings of the 9th Annual European Symposium on Algorithms*, pages 182–193. Springer-Verlag, 2001.
9. Lei Li, Jong Hyun Kim, and Michael S. Waterman. Haplotype reconstruction from SNP alignment. In *Proceedings of the seventh annual international conference on Computational molecular biology*, pages 207–216. ACM Press, 2003.

10. Lakshmi K. Matukumalli, John J. Grefenstette, David L. Hyten, Ik-Young Choi, Perry B. Cregan, and Curtis P. Van Tassell. Application of machine learning in SNP discovery. *BMC Bioinformatics*, 7:4, 2006.
11. Gene Myers. A dataset generator for whole genome shotgun sequencing. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 202–210. AAAI Press, 1999.
12. Alessandro Panconesi and Mauro Sozio. Fast hare: A fast heuristic for single individual SNP haplotype reconstruction. In *WABI*, pages 266–277, 2004.
13. Sachidanandam R. and al. A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms. In *Nature 409*, pages 928–933, February 2001.
14. Romeo Rizzi, Vineet Bafna, Sorin Istrail, and Giuseppe Lancia. Practical algorithms and fixed-parameter tractability for the single individual SNP haplotyping problem. In *Proceedings of the Second International Workshop on Algorithms in Bioinformatics*, pages 29–43. Springer-Verlag, 2002.
15. Rui-Sheng Wang, Ling-Yun Wu, Zhen-Ping Li, and Xiang-Sun Zhang. Haplotype reconstruction from snp fragments by minimum error correction. *Bioinformatics*, 21(10):2456–2462, 2005.
16. M.P. Weiner and T.J. Hudson. Introduction to SNPs: discovery of markers for disease. *Biotechniques*, Suppl, 2002.
17. Yu-Ying Zhao, Ling-Yun Wu, Ji-Hong Zhang, Rui-Sheng Wang, and Xiang-Sun Zhang. Haplotype assembly from aligned weighted snp fragments. *Computational Biology and Chemistry*, 29(4):281–287, 2005.