

Practical Algorithms and Fixed-Parameter Tractability for the Single Individual SNP Haplotyping Problem

Romeo Rizzi^{1,*}, Vineet Bafna², Sorin Istrail², and Giuseppe Lancia³

¹ Math. Dept., Università di Trento, 38050 Povo (Tn), Italy
`rrizzi@science.unitn.it`

² Celera Genomics, Rockville MD, USA,

`{Vineet.Bafna,Sorin.Istrail}@celera.com`

³ D.E.I., Università di Padova, 35100 Padova, Italy
`lancia@dei.unipd.it`

Abstract. Single nucleotide polymorphisms (SNPs) are the most frequent form of human genetic variation, of foremost importance for a variety of applications including medical diagnostic, phylogenies and drug design.

The complete SNPs sequence information from each of the two copies of a given chromosome in a diploid genome is called a *haplotype*. The Haplotyping Problem for a single individual is as follows: Given a set of fragments from one individual's DNA, find a maximally consistent pair of SNPs haplotypes (one per chromosome copy) by removing data “errors” related to sequencing errors, repeats, and paralogous recruitment. Two versions of the problem, i.e. the Minimum Fragment Removal (MFR) and the Minimum SNP Removal (MSR), are considered.

The Haplotyping Problem was introduced in [8], where it was proved that both MSR and MFR are polynomially solvable when each fragment covers a set of consecutive SNPs (i.e., it is a *gapless* fragment), and NP-hard in general. The original algorithms of [8] are of theoretical interest, but by no means practical. In fact, one relies on finding the maximum stable set in a perfect graph, and the other is a reduction to a network flow problem. Furthermore, the reduction does not work when there are fragments completely included in others, and neither algorithm can be generalized to deal with a bounded total number of holes in the data.

In this paper, we give the first practical algorithms for the Haplotyping Problem, based on Dynamic Programming. Our algorithms do not require the fragments to not include each other, and are polynomial for each constant k bounding the total number of holes in the data.

For m SNPs and n fragments, we give an $O(mn^{2k+2})$ algorithm for the MSR problem, and an $O(2^{2k}m^2n + 2^{3k}m^3)$ algorithm for the MFR problem, when each fragment has at most k holes. In particular, we obtain an $O(mn^2)$ algorithm for MSR and an $O(m^2n + m^3)$ algorithm for MFR on gapless fragments.

Finally, we prove that both MFR and MSR are APX-hard in general.

* Research partially done while enjoying hospitality at BRICS, Department of Computer Science, University of Aarhus, Denmark.

1 Introduction

With the sequencing of the human genome [12,7] has come the confirmation that all humans are almost identical at DNA level (99% and greater identity). Hence, small regions of differences must be responsible for the observed diversities at phenotype level. The smallest possible variation is at a single nucleotide, and is called *Single Nucleotide Polymorphism*, or SNP (pronounced “snip”). Broadly speaking, a polymorphism is a trait, common to everybody, whose value can be different but drawn in a limited range of possibilities, called *alleles*. A SNP is a specific nucleotide, placed in the middle of a DNA region which is otherwise identical for all of us, whose value varies within a population. In particular, each SNP shows a variability of only two alleles. These alleles can be different for different SNPs.

Recent studies have shown that SNPs are the predominant form of human variation [2] occurring, on average, every thousand bases. Their importance cannot be overestimated for therapeutic, diagnostic and forensic applications. Nowadays, there is a large amount of research going on in determining SNP sites in humans as well as other species, with a SNP consortium founded with the aim of designing a detailed SNP map for the human genome [11,6].

Since DNA of *diploid* organisms is organized in pairs of chromosomes, for each SNP one can either be *homozygous* (same allele on both chromosomes) or *heterozygous* (different alleles). The values of a set of SNPs on a particular chromosome copy define a *haplotype*. *Haplotyping* an individual consists in determining a pair of haplotypes, one for each copy of a given chromosome. The pair provides full information of the SNP fingerprint for that individual at the specific chromosome.

There exist different combinatorial versions of haplotyping problems. In particular, the problem of haplotyping a population (i.e., a set of individuals) has been extensively studied, under many objective functions [4,5,3], while haplotyping for a single individual has been studied in [8] and in [9].

Given complete DNA sequence, haplotyping an individual would consist of a trivial check of the value of some nucleotides. However, the complete DNA sequence is obtained by the assembly of smaller fragments, each of which can contain errors, and is very sensitive to repeats. Therefore, it is better to define the haplotyping problem considering as input data the fragments instead of the fully assembled sequence.

Computationally, the haplotyping problem calls for determining the “best” pair of haplotypes, which can be inferred from data which is possibly inconsistent and contradictory. The problem was formally defined in [8], where conditions are derived under which it results solvable in polynomial time and others for which it is NP-hard. We remark that both situations are likely to occur in real-life contexts, depending on the type of data available and the methodology used for sequencing. In this paper we improve on both the polynomial and hardness results of [8]. In particular, we describe practical effective algorithms based on Dynamic Programming, which are low-degree polynomial in the number of SNPs and fragments, and remain polynomial even if the fragments are allowed to skip

Table 1. A chromosome and the two haplotypes

Chrom. <i>c</i> , paternal:	atagggtccCtattttccaggcgCgtatacttcgacgggActata		
Chrom. <i>c</i> , maternal:	atagggtccGtattttccaggcgCgtatacttcgacgggTctata		
Haplotype 1 →	C	C	A
Haplotype 2 →	G	C	T

some SNPs (up to a fixed maximum). As for the complexity results, we show that the problems are not just NP-hard, but in fact APX-hard.

Despite the biological origin of the problem, the model turns out to be purely combinatorial, and has many nice mathematical properties. The basic biological motivations behind the problem are provided in Section 2. The mathematical model of the problems, together with the notation and two useful reductions, are described in Section 3. In Section 4, we introduce a suitable bipartite labeled graph, used to characterize the problems and to give an APX-hardness result for their general versions. In Section 5 we describe Dynamic Programming-based polynomial algorithms for the gapless versions of the problem, while in Section 6 we extend these results to bounded-length gaps.

2 SNPs and Haplotypes

The process of passing from the sequence of nucleotides in a DNA molecule to a string over the DNA alphabet is called *sequencing*. A *sequencer* is a machine that is fed some DNA and whose output is a string of As, Ts, Cs and Gs. To each letter, the sequencer attaches a value (*confidence level*) which represents, essentially, the probability that the letter has been correctly read.

The main problem with sequencing is that the technology is not powerful enough to sequence a long DNA molecule, which must therefore first be cloned into many copies, and then be broken, at random, into several pieces (called *fragments*), of a few hundred nucleotides each, that are individually fed to a sequencer. The cloning phase is necessary so that the fragments can have nonempty overlap. From the overlap of two fragments one infers a longer fragment, and so on, until the original DNA sequence has been reconstructed. This is, in essence, the principle of *Shotgun Sequencing* [13], in which the fragments are *assembled* back into the original sequence by using sophisticated algorithms. The assembly phase is complicated from the fact that in a genome there exist many regions with identical content (*repeats*) scattered all around, which may fool the assembler into thinking that they are all copies of a same, unique, region. The situation is complicated further from the fact that *diploid* genomes are organized into pairs of chromosomes (a paternal and a maternal copy) which may have identical or nearly identical content, a situation that makes the assembly process even harder.

To partly overcome these difficulties, the fragments used in shotgun sequencing sometimes have some extra information attached. In fact, they are obtained via a process that generates pairs (called *mate pairs*) of fragments instead than

individual ones, with a fairly precise estimate of the distance between them. These pairs are guaranteed to come from the same copy of a chromosome, and there is a good chance that, even if one of them comes from a repeat region, the other does not.

A *Single Nucleotide Polymorphism*, or SNP, is a position in a genome at which, within a population, some individuals have a certain base while the others have a different one. In this sense, that nucleotide is polymorphic, from which the name. For each SNP, an individual is *homozygous* if the SNP has the same allele on both chromosome copies, and otherwise the individual is *heterozygous*. The values of a set of SNPs on a particular chromosome copy define a *haplotype*. In Figure 1 we give a simplistic example of a chromosome with three SNP sites. The individual is heterozygous at SNPs 1 and 3 and homozygous at SNP 2. The haplotypes are CCA and GCT.

The *Haplotyping Problem* consists in determining a pair of haplotypes, one for each copy of a given chromosome, from some input genomic data. Given the assembly output (i.e., a fully sequenced genome) haplotyping would simply consist in checking the value of some specific sites. However, there are unavoidable errors, some due to the assembler, some to the sequencer, that complicate the problem and make it necessary to proceed in a different way. One problem is due to repeat regions and “paralogous recruitment” [9]. In practice, fragments with high similarity are merged together, even if they really come from different chromosome copies, and the assembler tends to reconstruct a single copy of each chromosome. Note that in these cases, heterozygous SNP sites could be used to correct the assembly and segregate two distinct copies of the similar regions. Another problem is related to the quality of the reads. For these reasons, the haplotyping problem has been recently formalized as a combinatorial problem, defined not over the assembly output, but over the original set of fragments. The framework for this problem was introduced in [8]. The data consists of small, overlapping fragments, which can come from either one of two chromosome copies. Further, e.g. in shotgun sequencing, there may be pairs of fragments known to come from the same chromosome copy and to have a given distance between them. Because of unavoidable errors, under a general parsimony principle, the basic problem is the following:

- *Given a set of fragments obtained by DNA sequencing from the two copies of a chromosome, find the smallest number of errors so that there exist two haplotypes compatible with all the (corrected) fragments observed.*

Depending on the errors considered, different combinatorial problems have been defined in the literature. “Bad” fragments can be due either to contaminants (i.e. DNA coming from a different organism than the actual target) or to read errors. An alternative point of view assigns the errors to the SNPs, i.e. a “bad” SNP is a SNP for which some fragments contain read errors. Correspondingly, we have the following optimization problems: “*Find the minimum number of fragments to ignore*” or “*Find the minimum number of SNPs to ignore*”, such that “*the (corrected) data is consistent with the existence of two haplotypes. Find such haplotypes.*”

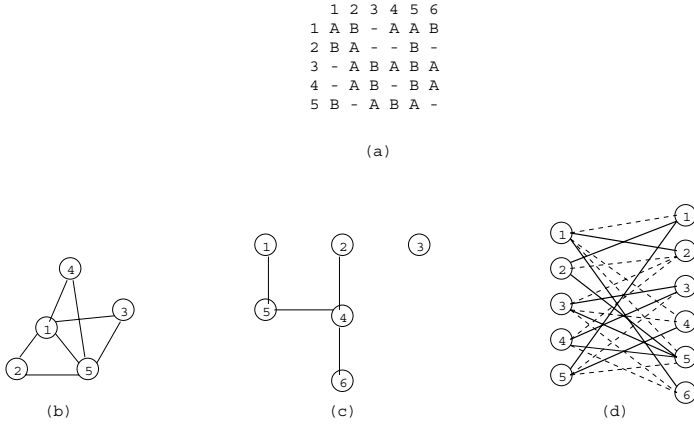


Fig. 1. (a) A SNP matrix. (b) The Fragment conflict graph $G_{\mathcal{F}}(M)$. (c) The SNP conflict graph $G_{\mathcal{S}}(M)$. (d) The labeled bipartite graph (G, ℓ)

3 Terminology and Notation

Let $\mathcal{S} = \{1, \dots, n\}$ be a set of SNPs and $\mathcal{F} = \{1, \dots, m\}$ be a set of fragments (where, for each fragment, only the nucleotides at positions corresponding to some SNP are considered). Each SNP is covered by some of the fragments, and can take only two values. The actual values (nucleotides) are irrelevant to the combinatorics of the problem and hence we will denote, for each SNP, by A and B the two values it can take. Given any ordering of the SNPs (e.g., the natural one, induced by their physical location on the chromosome), the data can also be represented by an $m \times n$ matrix over the alphabet $\{A, B, -\}$, which we call the *SNP matrix* (read “snip matrix”), defined in the obvious way. The symbol $-$ appears in all cells $M[f, s]$ for which a fragment f does not cover a SNP s , and it is called a *hole*.

For a SNP s , two fragments f and g are said to *conflict on s* if $M[f, s] = A$ and $M[g, s] = B$ or vice-versa. Two fragments f and g are said to *conflict* if there exists a SNP s such that they conflict on s , otherwise f and g are said to *agree*. A SNP matrix M is called *error-free* if we can partition the rows (fragments) into two classes of non-conflicting fragments.

Given a SNP matrix M , the *fragment conflict graph* is the graph $G_{\mathcal{F}}(M) = (\mathcal{F}, E_{\mathcal{F}})$ with an edge for each pair of conflicting fragments (see Figure 1(a) and (b)). Note that if M is error-free, $G_{\mathcal{F}}(M)$ is a bipartite graph, since each haplotype defines a shore of $G_{\mathcal{F}}(M)$, made of all the fragments coming from that haplotype. Conversely, if $G_{\mathcal{F}}(M)$ is bipartite, with shores H_1 and H_2 , all the fragments in H_1 can be merged into one haplotype and similarly for H_2 . Hence, M is error-free if and only if $G_{\mathcal{F}}(M)$ is bipartite.

The fundamental underlying problem in SNP haplotyping is determining an optimal set of changes to M (e.g., row and/or column- deletion) so that M becomes error-free. Given a matrix M , and where X is any set of rows or columns

of M , we denote by $M \setminus X$ the matrix obtained from M by dropping the rows or columns in X . In this work, we will consider the following problems.

MSR Minimum SNP Removal - Find a minimum number of columns (SNPs) whose removal makes M error-free;

MFR Minimum Fragment Removal - Find a minimum number of rows (fragments) whose removal makes M error-free.

For better readability, from now on we will refer to “a matrix M ” instead of “a SNP matrix M ”, unless a possible confusion arises with another type of matrix. For a problem $\Pi \in \{\text{MSR}, \text{MFR}\}$ on input a matrix M , we will denote by $\Pi(M)$ the value of an optimal solution.

The following two reductions can be used to remove redundant data from the input, and hence to clean the structure of the problems.

We start by considering the minimum number of columns (SNPs) whose removal makes M error-free. We have the following proposition:

Proposition 1 ($\mathcal{S}$ -reduction) *Let M' be the matrix obtained from M by dropping those columns where no **A**'s or no **B**'s occur. Clearly, $\text{MSR}(M') \leq \text{MSR}(M)$. Let X be any set of SNPs such that $M' \setminus X$ is error-free. Then also $M \setminus X$ is error-free.*

Essentially, Proposition 1 says that when solving the problem we can simply concentrate our attention to M' , the other columns being inessential. Matrix M' so obtained is called $\mathcal{S}$ -reduced. When M is error-free, then we say that two fragments f and g are *allies* (*enemies*) when they must be in the same class (in separate classes) for every partition of the rows of M into two classes of non-conflicting fragments.

Now, on an $\mathcal{S}$ -reduced M' , we have the following structure for solutions.

Lemma 2 (“o con noi o contro di noi”)¹ *Let M be an $\mathcal{S}$ -reduced matrix. Let X be any set of SNPs whose removal makes M error-free. Let $f, g \in \mathcal{F}$ be any two fragments. Consider any SNP $s \in \mathcal{S} \setminus X$ such that $\{M[f, s], M[g, s]\} \subseteq \{\mathbf{A}, \mathbf{B}\}$. Then, if $M[f, s] \neq M[g, s]$ then f and g are enemies, otherwise, they are allies.*

Proof: The first part is obvious. As for the second, assume, e.g., $M[f, s] = M[g, s] = \mathbf{A}$. Then, since M is $\mathcal{S}$ -reduced, there exists a third fragment h such that $M[h, s] = \mathbf{B}$. □

We also have a similar reduction which applies to rows.

Proposition 3 ($\mathcal{F}$ -reduction) *Let M' be the matrix obtained from M by dropping those rows which conflict with at most one other row. Clearly, $\text{MSR}(M') \leq \text{MSR}(M)$. Let X be any set of SNPs whose removal makes M' error-free. Then the removal of X makes also M error-free.*

¹ Literal translation: either with us or against us. Meaning: you have to choose, either be friend or enemy

We can hence assume that every row conflicts with at least two other rows, simply by dropping those rows which conflict with at most one row. Matrix M' so obtained is called $\mathcal{F}$ -reduced.

We now proceed to check that the two reductions just introduced for MSR are also valid for MFR (they are valid for MER as well). So, consider the minimum number of rows (fragments) whose removal makes M error-free. The following propositions are easy to prove

Proposition 4 ($\mathcal{S}$ -reduction, again) *Let M' be the matrix obtained from M by dropping those columns where only A's or only B's occur. Clearly, $\text{MFR}(M') \leq \text{MFR}(M)$. Let X be any set of rows whose removal makes M' error-free. Then also $M \setminus X$ is error-free.*

Proposition 5 ($\mathcal{F}$ -reduction, again) *Let M' be the matrix obtained from M by dropping those rows which conflict with at most one single other row. Clearly, $\text{MFR}(M') \leq \text{MFR}(M)$. Let X be any set of rows whose removal makes M' error-free. Then also $M \setminus X$ is error-free.*

3.1 Old and New Results

We define a *gapless* fragment i as one for which the As and Bs appear consecutively, with no - s between them, in row i of M . In general, a *gap* is a maximal run of consecutive holes between two non-hole symbols. As an example, --ABABBBA----, is a gapless fragment, while there are 2 gaps in --AB---B--AB-. The *length* of a *gap* is the number of holes it contains (so, the above second example, has a total gap length of $5 = 3 + 2$). The *body* of a fragment extends from the leftmost non-hole to the rightmost non-hole (e.g., the body of ---ABB-B--AB- is ABB-B--AB).

In haplotyping, there can be gaps for mainly two reasons:

1. *thresholding of low-quality reads.* This happens if the sequencer cannot call a SNP A or B with enough confidence; then, no call is made, and the position is marked with -.
2. *mate-pairing in shotgun sequencing.* One pair of mates are two fragments coming from the same chromosome copy, with a given distance between them. Hence, they are logically equivalent to a single fragment with one gap.

We call a matrix *gapless* if all fragments are gapless. We say that a matrix *has the consecutive-1 property* (is C1P) if the columns can be rearranged so as to obtain a gapless matrix. Note that determining if such a rearrangement exists is a well-known polynomial problem [1]. As a consequence, all polynomial results for gapless matrices can be readily extended to polynomial results for C1P matrices. We note also that the consecutive-1 property is conserved under both $\mathcal{S}$ -reduction and $\mathcal{F}$ -reduction.

We now briefly recall the main results that were obtained in [8] for gapless matrices and in Sections 5.1 and 5.2 we go on to improve them to more practical and effective algorithms. In a nutshell, in [8] it is shown that for gapless matrices

the problems are polynomial, while, in general, they are NP-hard. Note that these cases are both likely to occur in real-life applications. For instance, an EST (Expressed Tagged Sequence) is a short DNA fragment with no gaps, fully sequenced. When the input consists only of ESTs, the matrix M is C1P (i.e., has the consecutive 1 property). On the other hand, when also mate pairs are used, the matrix is not necessarily C1P.

Let M be an $\mathcal{S}$ -reduced matrix. Two SNPs s and t are said to *be in conflict* if there exists fragments f and g such that $M[f, a]$, $M[g, a]$, $M[f, c]$, $M[g, c] \neq -$ and the boolean value $(M[f, a] = M[g, a])$ is the negation of $(M[f, c] = M[g, c])$. In other words, the 2×2 submatrix of M defined by rows f and g and columns s and t has 3 symbols of one type (A or B) and one of the opposite (B or A respectively). Given a matrix M , the *SNP conflict graph* is the graph $G_{\mathcal{S}}(M) = (\mathcal{S}, E_{\mathcal{S}})$, with an edge for each pair of SNPs in conflict (see Figure 1(c)).

The following theorems are proved in [8]

Theorem 6 *Let M be a gapless matrix. Then M is error-free if and only if $G_{\mathcal{S}}(M)$ is a stable set.*

Theorem 7 *If M is a gapless matrix, then $G_{\mathcal{S}}(M)$ is a weakly triangulated graph.*

From theorems 6 and 7 it followed that MSR is polynomial for a gapless matrix, since it amounts to finding the largest stable set in a perfect graph. The main result in [8] was to show that the problem is polynomial, and little attention was paid to the fact that the running times of algorithms based on the perfectness of graphs are bounded by too high-degree polynomials for practical applications (let apart the problem of actually coding these algorithms). Similarly, in [8] a point was made that, for gapless data, also MFR is polynomial, which was shown via an expensive reduction to a network flow problem. Furthermore, the result is only valid when no fragment is contained in another. In the next two sections, we set up for quicker and much simpler algorithms for both MFR and MSR. We also manage to get rid of the assumption that no fragment is contained in another. The main results we establish in this paper are the following:

Theorem 8 *There is an $O(mn^{2k+2})$ polynomial time algorithm for MSR problem on matrices in which each fragment has total gap length at most k .*

Corollary 9 *There is an $O(mn^2)$ polynomial time algorithm for MSR problem on matrices which have the consecutive-1 property.*

Theorem 10 *There is an $O(2^{2k}m^2n + 2^{3k}m^3)$ polynomial time algorithm for MFR problem on matrices in which each fragment has total gap length at most k .*

Corollary 11 *There is an $O(m^2n + m^3)$ polynomial time algorithm for MFR problem on matrices which have the consecutive-1 property.*

Theorem 12 *The problems MFR and MSR are, in general, APX-hard.*

4 The Data as Labeled Bipartite Graph

We assume that we are working on an $\mathcal{S}$ -reduced matrix M . To M we associate a labeled graph as follows. Let $G = (\mathcal{F}, \mathcal{S}; E)$ be a bipartite graph on color classes $\mathcal{F}$ and $\mathcal{S}$ and with edge set $E := \{sf : s \in \mathcal{S}, f \in \mathcal{F}, M[s, f] \neq -\}$. When $M[s, f] = \mathbf{A}$ we label sf *even*, and set $\ell(sf) = 0$. When $M[s, f] = \mathbf{B}$ we label sf *odd*, and set $\ell(sf) = 1$ (see Figure 1(d), where even edges are dashed and odd edges are thick). An edge set $F \subseteq E$ is called ℓ -*odd* if it contains an odd number of odd edges, and ℓ -*even* otherwise. Indeed, we can extend our labeling to edge sets as $\ell(F) := \left(\sum_{f \in F} \ell(f) \right)_{\text{mod } 2}$. We say that the pair (G, ℓ) is ℓ -*bipartite* if and only if it contains no ℓ -odd cycle (regard a cycle as an edge set). By the “o con noi o contro di noi” Lemma 1, we have the following consequence:

Proposition 13 *The matrix M is error-free if and only if (G, ℓ) is ℓ -bipartite.*

We now give a formal proof of the above proposition in a more general form.

Lemma 14 *Let X be a set of columns. If M is $\mathcal{S}$ -reduced and $M \setminus X$ is error-free, then $(G \setminus X, \ell)$ is ℓ -bipartite.*

Proof: Indeed, let $C = f_1, s_1, \dots, f_k, s_k$ be any cycle in $G \setminus X$. Define $\ell(\delta_C(s_i)) = \ell(f_i s_i) + \ell(s_i f_{i+1})$, so that $\ell(C) = \sum_{i=1}^k \ell(\delta_C(s_i))$. Consider a partition of $\mathcal{F}$ into two haplotypes. By Lemma 1, whenever $\ell(\delta_C(s_i))$ is even, f_i and f_{i+1} are in the same haplotype, while when it is odd, they are in different haplotypes. But, along the cycle $f_1, f_2, \dots, f_k$, the number of haplotype switches must be even (like jumping forth and back between the two sides of a river but eventually returning to the starting side). $\square$

Now we go for the converse.

Lemma 15 *If (G, ℓ) is ℓ -bipartite then M (not necessarily reduced) is error-free.*

Proof: By definition, the fragment conflict graph $G_{\mathcal{F}}(M) = (\mathcal{F}, E_{\mathcal{F}})$ is such that $uv \in E_{\mathcal{F}}$ if there is an s such that us and sv are in (G, ℓ) and $\ell(us) + \ell(sv) = 1$. Hence, to each cycle C in $G_{\mathcal{F}}(M)$ corresponds a cycle C' in (G, ℓ) , and C has an even number of edges if and only if C' is ℓ -even. So, $G_{\mathcal{F}}(M)$ is bipartite and hence M is error-free. $\square$

We finally prove Theorem 12 of Section 3.1.

Theorem 16 *The problems MFR and MSR are APX-hard.*

Proof: Given a graph G , the problem (MINEDGEBIPARTIZER) of finding a minimum cardinality set of edges F such that $G \setminus F$ is bipartite, and the problem (MINNODEBIPARTIZER) of finding a minimum cardinality set of nodes Z such that $G \setminus Z$ is bipartite, are known to be APX-hard [10]. Moreover, the same problems are not known to be in APX. We give simple L -reductions from these two problems to MFR and MSR, hence showing the APX-hardness of MFR and

MSR, but also that finding a constant ratio approximation algorithm for any of these problems in general is somewhat of a challenge.

Given an input graph G as instance for either MINEDGEBIPARTIZER or MINNODEBIPARTIZER, we subdivide each edge into two edges in series. (The resulting graph G' is clearly bipartite, we call SNPs the new nodes and fragments the old ones). Now, of the two edges incident with every SNP, we declare one to be odd and one to be even. Clearly, solving or approximating MSR on (G', ℓ) amount to solving or approximating (within exactly the same approximation) MINEDGEBIPARTIZER on G . Moreover, solving or approximating MFR on (G', ℓ) amounts to solving or approximating (within exactly the same approximation) MINNODEBIPARTIZER on G . $\square$

5 Polynomial Algorithms for the Gapless Case

In this section we prove Corollaries 9 and 11 of Section 3.1. Although they would follow from the more general theorems for matrices with bounded length gaps (described in Section 6) we prove them here directly because it is didactically better to do so. In fact, the results of Section 6 will be obtained as generalizations of the ideas described in this section.

5.1 MSR: A Dynamic Programming $O(mn^2)$ Algorithm

In this section we propose a dynamic programming approach for the solution of MSR. The resulting algorithm can be coded as to take $O(mn^2)$ time. In the following, we assume M to be $\mathcal{S}$ -reduced.

It is easier to understand our dynamic program if we state it for the complementary problem of MSR, i.e., find the maximum number of SNPs that can be kept so that M becomes error-free. Clearly, if k is the largest number of SNPs that we can keep, then $n - k$ is the smallest number of SNPs to remove.

For $j \leq n$ (with $j \geq 0$), we define $K[j]$ as the maximum number of columns that can be kept to make M error-free, under the condition that j is the rightmost column kept (if all columns are removed then $j = 0$, and $K[0] = 0$).

Once all the $K[j]$ are known, the solution to the problem is given by

$$\max_{j \in \{0, \dots, n\}} K[j].$$

For every j , we define $\text{OK}(j)$ as the set of those i with $i < j$ such that columns i and j do not conflict. We assume that 0 belongs to $\text{OK}(j)$ for every j . Now, for every j ,

$$K[j] := 1 + \max_{i \in \text{OK}(j)} K[i] \tag{1}$$

where Equation (1) is correct by the following easily proven fact.

Lemma 17 *Let M be a gapless $\mathcal{S}$ -reduced matrix. Consider columns $a < b < c \in \mathcal{S}$. If a is not in conflict with b and b is not in conflict with c , then a is not in conflict with c .*

Proof: Assume SNPs a and c to be conflicting, that is, there exist fragments f and g such that $M[f, a]$, $M[g, a]$, $M[f, c]$, $M[g, c] \neq -$ and the boolean value $(M[f, a] = M[g, a])$ is the negation of $(M[f, c] = M[g, c])$. Since $a < b < c$, then $M[f, b], M[g, b] \neq -$ since M is gapless. Therefore, $(M[f, b] = M[g, b])$ is either the negation of $(M[f, a] = M[g, a])$ or the negation of $(M[f, c] = M[g, c])$. That is, b is either conflicting with a or with c . $\square$

Note that for computing the entries $K[j]$ we only need to know the sets $\text{OK}(j)$. Note that $\text{OK}(j)$ is the set of those $i < j$ which are neighbors of j in the SNP-conflict graph $G_S(M)$. Since determining if two SNPs are in conflict can be done in time $O(m)$, the cost of creating the $\text{OK}(j)$ is $O(mn^2)$. This dominates the cost $O(n^2)$ of solving all equations (1).

5.2 MFR: A Dynamic Programming $O(m^2n + m^3)$ Algorithm

In this section we propose a dynamic programming approach for the solution of MFR. We remark that, contrary to the approach suggested in [8], nested fragments will not be a problem. The resulting algorithm can be coded as to take $O(m^2n + m^3)$ time.

Given a row f of M we denote by $l(f)$ the index of the leftmost SNP s such that $M[f, s] \neq -$ and by $r(f)$ the index of the rightmost SNP s such that $M[f, s] \neq -$. In other words, the body of the fragment f is all contained between the SNPs $l(f)$ and $r(f)$. We assume that the rows of M are ordered so that $l(i) \leq l(j)$ whenever $i < j$. For every $i \in \{1, \dots, m\}$, let M_i be the matrix made up by the first i rows of M . For $h, k \leq i$ (with $h, k \geq -1$) such that $r(h) \leq r(k)$, we define $D[h, k; i]$ as the minimum number of rows to remove to make M_i error-free, under the condition that

- row k is not removed, and among the non-removed rows maximizes $r(k)$;
- row h is not removed and goes into the opposite haplotype as k , and among such rows maximizes $r(h)$.

(If all rows are removed then $h = -1, k = 0$, and $D[-1, 0; i] := i$. Rows -1 and 0 are all $-$, that is, empty).

Once all the $D[h, k; i]$ are known, the solution to the MSR problem is given by

$$\min_{h, k : r(h) \leq r(k)} D[h, k; m].$$

Clearly, for every i , and for every $h, k < i$ with $r(h) \leq r(k)$,

$$D[h, k; i] := \begin{cases} D[h, k; i-1] & \text{if } r(i) \leq r(k) \text{ and rows } i \text{ and } k \text{ agree} \\ D[h, k; i-1] & \text{if } r(i) \leq r(h) \text{ and rows } i \text{ and } h \text{ agree} \\ D[h, k; i-1] + 1 & \text{otherwise,} \end{cases} \quad (2)$$

where Equation (2), as well as Equation (3) and Equation (4), finds explanation into the following easily proven fact.

Lemma 18 *Consider rows $a, b, c \in \mathcal{F}$. Assume $a, b < c$ and $r(a) \leq r(b)$. If a agrees with b and b agrees with c , then a agrees with c .*

For every i , we define $\text{OK}(i)$ as the set of those j with $j < i$ such that rows i and j agree. We assume that $0, -1$ belong to $\text{OK}(i)$ for every i . (Just think to append a pair of all “-” rows at the top of M).

Now, for every i , and for every $h < i$ with $r(h) \leq r(i)$,

$$D[h, i; i] := \min_{j \in \text{OK}(i), j \neq h, r(j) \leq r(i)} \begin{cases} D[j, h; i-1] & \text{if } r(h) \geq r(j) \\ D[h, j; i-1] & \text{if } r(h) < r(j) \end{cases} \quad (3)$$

Finally, for every i , and for every $k < i$ with $r(k) \geq r(i)$,

$$D[i, k; i] := \min_{j \in \text{OK}(i), j \neq k, r(j) \leq r(i)} D[j, k; i-1]. \quad (4)$$

Note that for computing the entries $D[h, k; i]$ we only need to know the sets $\text{OK}(i)$. The cost of creating the $\text{OK}(i)$ data structure (done in a first phase) is $O(m^2n)$. The cost of computing the entries $D[h, k; i]$ (done in a second phase) is $O(m^3)$, since it can be seen as the cost of computing the $O(m^3)$ entries $D[h, k; i]$ by using Equation (2) (costs $O(1)$ each) plus the cost of computing the $O(m^2)$ entries $D[h, i; i]$ and $D[i, k; i]$ by using Equations (3) and (4) (costs $O(m)$ each).

6 Dealing with Gaps

In this section we propose a practical approach to deal with any number of gaps, when the body of each fragment does not contain many holes. For the remainder of this section, let k be a constant such that the body of each fragment in the input instance contains at most k holes. We will derive dynamic programming-based polynomial (for a constant k) algorithms for both MFR and MSR, proving Theorems 8 and 10 of Section 3.

6.1 MSR: An $O(mn^{2k+2})$ Algorithm

We modify the basic dynamic programming approach for MSR introduced in Section 5.1. More precisely, the new dynamic programming algorithm for MSR will now consider $2k$ columns of history. We remind the reader that we show how to determine the largest number of SNPs that can be kept to make M error-free, which is equivalent to solving MSR. The resulting algorithm can be coded as to take $O(mn^{2k+2})$ time. In the following, we assume M to be $\mathcal{S}$ -reduced.

We say that SNPs $s_1, \dots, s_t$ are *consistent* when no two of them conflict. For consistent $j_1 < \dots < j_{2k+1} \leq i$ (with $j_1 \geq -2k$), we define $K[j_1, \dots, j_{2k+1}]$ as the maximum number of columns to keep to make M error-free, under the condition that $j_1, \dots, j_{2k+1}$ are the $2k+1$ rightmost columns kept (if all columns are removed then $j_{2k+1} = 0$, and $K[-2k, \dots, -1, 0] = 0$).

Once all the $K[\dots]$ are known the solution to the problem is given by

$$\max_{-2k < j_1 < \dots < j_{2k+1} \leq n} K[j_1, \dots, j_{2k+1}].$$

For $i \leq n$ and consistent $j_1 < \dots < j_{2k} < i$ we define $\text{OK}(j_1, \dots, j_{2k}, i)$ as the set of those $j < j_1$ such that columns $j, j_1 < \dots < j_{2k}, i$ are consistent.

Now, for every consistent $j_1 < \dots < j_{2k} < i$,

$$K[j_1, \dots, j_{2k}, i] := 1 + \max_{j \in \text{OK}(j_1, \dots, j_{2k}, i)} K[j, j_1, \dots, j_{2k}] \quad (5)$$

where Equation (5) is correct by the following easily proven fact.

Lemma 19 *Let M be an $\mathcal{S}$ -reduced matrix where each fragment contains at most k holes. Consider columns $a < j_1, \dots, j_{2k} < c \in \mathcal{S}$. Assume columns $a, j_1, \dots, j_{2k}$ are consistent. Assume further columns $j_1, \dots, j_{2k}, c$ are consistent. Then, columns $a, j_1, \dots, j_{2k}, c$ are consistent.*

Proof: Assume on the contrary that a and c conflict. Let f and g be fragments such that $M[f, a], M[g, a], M[f, c], M[g, c] \neq -$ and the boolean value $(M[f, a] = M[g, a])$ is the negation of $(M[f, c] = M[g, c])$.

Since each row has at most k holes, as a consequence, on $2k+1$ columns, any two rows must have a common non-hole or one of the considered columns is out of the body of one of the two rows. Since a and c are both in the body of f then $j_1, \dots, j_{2k}$ are all in the body of f . Similarly, $j_1, \dots, j_{2k}$ are all in the body of g . Hence, let $b \in \{j_1, \dots, j_{2k}\}$ such that $M[f, b], M[g, b] \neq -$. Since $a < b < c$, then $M[f, b], M[g, b] \neq -$ since M is gapless. Therefore, $(M[f, b] = M[g, b])$ is either the negation of $(M[f, a] = M[g, a])$ or the negation of $(M[f, c] = M[g, c])$. That is, b is either conflicting with a or with c . $\square$

Note that for computing the entries $K[\dots; i]$ we only need to know the sets $\text{OK}(j_1, \dots, j_{2k}, i)$. The cost of creating the $\text{OK}(j_1, \dots, j_{2k}, i)$ (done in a first phase) is $O(mn^{2k+2})$, which dominates the cost $O(n^{2k+2})$ of solving all equations (5).

6.2 MFR: An $O(2^{2k}nm^2 + 2^{3k}m^3)$ Algorithm

We show how to extend the dynamic programming approach given in Section 5.2 to solve gapless problem instances with holes in $O(2^{2k}nm^2 + 2^{3k}m^3)$ time. We point out that the form of the necessarily-exponential dependence on k is a very good one, i.e., it shows the fixed-parameter tractability of the problem. The memory requirement is $2^{2k}m^3$.

Let f be a fragment and let $x \in \{A, B\}^k$. We denote by $f[x]$ the fragment obtained from f by filling in the holes one by one, using the first characters in x . Since we assumed that the body of each fragment in our input instance contains at most k holes, the characters in x will always suffice to fill in all the holes of f . Given $x_1, x_2 \in \{A, B\}^k$ and two rows f_1, f_2 of matrix M , we denote by $M[f_1[x_1], f_2[x_2]]$ the matrix obtained from M by substituting f_1 with $f_1[x_1]$ and f_2 with $f_2[x_2]$. The following consideration suggests how to extend the dynamic programming algorithm given in Section 5.2 to the case with holes:

Proposition 20 *Let $F_1 = \{f_1^1, \dots, f_1^p\}, F_2 = \{f_2^1, \dots, f_2^q\}$ be sets of fragments in M such that any two fragments in F_i ($i = 1, 2$) agree. Then, for every $i \leq p$*

and $j \leq q$ we can give $x_1^i, x_2^j \in \{A, B\}^k$ such that $F_1' = \{f_1^1, \dots, f_1^i[x_1^i], \dots, f_1^p\}$ and $F_2' = \{f_2^1, \dots, f_2^j[x_2^j], \dots, f_2^q\}$ would still be both without conflicts.

We assume that the rows of M are ordered so that $l(i) \leq l(j)$ whenever $i < j$. For every $i \in \{1, \dots, m\}$, let M_i be the matrix made up by the first i rows of M . For $h, k \leq i$ (with $h, k \geq -1$) such that $r(h) \leq r(k)$, and for $x, y \in \{A, B\}^k$ we define $D[h, x; k, y; i]$ as the minimum number of rows to remove to make $M_i[h[x], k[y]]$ error-free, under the condition that

- row $k[y]$ is not removed, and among the non-removed rows maximizes $r(k)$;
- row $h[x]$ is not removed and goes into the opposite haplotype as $k[y]$, and among such rows maximizes $r(h)$.

(If all rows are removed then $h = -1, k = 0$, and $D[-1, x; 0, y; i] = i$ for all $x, y \in \{A, B\}^k$.)

Once all the $D[h, x; k, y; i]$ are known, the solution to the problem is given by

$$\min_{x, y \in \{A, B\}^k; h, k : r(h) \leq r(k)} D[h, x; k, y; m].$$

Clearly, for every i , and for every $h, k < i$ with $r(h) \leq r(k)$, and for every $x, y \in \{A, B\}^k$,

$$D[h, x; k, y; i] := \begin{cases} D[h, x; k, y; i-1] & \text{if } r(i) \leq r(k) \text{ and rows } i \\ & \text{and } k[y] \text{ agree} \\ D[h, x; k, y; i-1] & \text{if } r(i) \leq r(h) \text{ and rows } i \\ & \text{and } h[x] \text{ agree} \\ D[h, x; k, y; i-1] + 1 & \text{otherwise} \end{cases} \quad (6)$$

For every fragment i and for every $x \in \{A, B\}^k$ we define $\text{OK}(i, x)$ as the set of those pairs (j, y) such that j is a fragment with $j < i$ and $y \in \{A, B\}^k$ such that rows $i[x]$ and $j[y]$ agree. Now, for every i , and for every $h < i$ with $r(h) \leq r(i)$, and for every $x_i, x_h \in \{A, B\}^k$,

$$D[h, x_h; i, x_i; i] := \min_{(j, x_j) \in \text{OK}(i, x_i), j \neq h, r(j) \leq r(i)} \begin{cases} D[j, x_j; h, x_h; i-1] & \text{if } r(h) \geq r(j) \\ D[h, x_h; j, x_j; i-1] & \text{if } r(h) < r(j) \end{cases} \quad (7)$$

Finally, for every i , and for every $k < i$ with $r(k) \geq r(i)$, and for every $x_i, x_k \in \{A, B\}^k$,

$$D[i, x_i; k, x_k; i] := \min_{(j, x_j) \in \text{OK}(i, x_i), j \neq k, r(j) \leq r(i)} D[j, x_j; k, x_k; i-1]. \quad (8)$$

Note that for computing the entries $D[h, x; k, y; i]$ we only need to know the sets $\text{OK}(i)$. The cost of creating the $\text{OK}(i, x)$ data structure (done in a first

phase) is $O(2^{2k}nm^2)$. The cost of computing the entries $D[h, x; k, y; i]$ (done in a second phase) is $O(2^{3k}m^3)$, since it can be seen as the cost of computing the $O(2^{2k}m^3)$ entries $D[h, x; k, y; i]$ with $h < k < i$ by using Equation (6) (costs $O(1)$ each) plus the cost of computing the $O(2^{2k}m^2)$ entries $D[h, x; i, y; i]$ by using Equations (7) and (8) (costs $O(2^k m)$ each).

References

1. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, intervals graphs and graph planarity testing using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.
2. A. Chakravarti. It's raining SNP, hallelujah? *Nature Genetics*, 19:216–217, 1998.
3. A. Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Molecular Biology Evolution*, 7:111–122, 1990.
4. D. Gusfield. A practical algorithm for optimal inference of haplotypes from diploid populations. In R. Altman, T.L. Bailey, P. Bourne, M. Gribskov, T. Lengauer, I.N. Shindyalov, L.F. Ten Eyck, and H. Weissig, editors, *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, pages 183–189, Menlo Park, CA, 2000. AAAI Press.
5. D. Gusfield. Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In G. Myers, S. Hannenhalli, S. Istrail, P. Pevzner, and M. Watermand, editors, *Proceedings of the Sixth Annual International Conference on Computational Biology*, pages 166–175, New York, NY, 2002. ACM Press.
6. L. Helmuth. Genome research: Map of the human genome 3.0. *Science*, 293(5530):583–585, 2001.
7. International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.
8. G. Lancia, V. Bafna, S. Istrail, R. Lippert, and R. Schwartz. SNPs problems, complexity and algorithms. In *Proceedings of Annual European Symposium on Algorithms (ESA)*, volume 2161 of *Lecture Notes in Computer Science*, pages 182–193. Springer, 2001.
9. R. Lippert, R. Schwartz, G. Lancia, and S. Istrail. Algorithmic strategies for the SNPs haplotype assembly problem. *Briefings in Bioinformatics*, 3(1):23–31, 2002.
10. C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In *Proceedings of 20th Int. Colloquium on Automata, Languages and Programming*, pages 40–51. Springer-Verlag, 1994.
11. E. Marshall. Drug firms to create public database of genetic mutations. *Science Magazine*, 284(5413):406–407, 1999.
12. J.C. Venter *et al.* The sequence of the human genome. *Science*, 291:1304–1351, 2001.
13. J. Weber and E. Myers. Human whole genome shotgun sequencing. *Genome Research*, 7:401–409, 1997.