Minimum Mosaic Inference of a Set of Recombinants

Guillaume Blin, Romeo Rizzi, Florian Sikora, Stéphane Vialette

Minimum Mosaic Inference of a Set of Recombinants

Guillaume Blin[3], Romeo Rizzi[1], Florian Sikora[2,3], and Stéphane Vialette[3]

[1] *DIMI, Università di Udine, Italy.* `romeo.rizzi@uniud.it`
[2] *Lehrstuhl für Bioinformatik, Friedrich-Schiller Universität Jena, Ernst-Abbe Platz 2, 00743 Jena, Germany.* `florian.sikora@uni-jena.de`
[3] *Université Paris-Est, LIGM - UMR CNRS 8049, 77454 Marne-la-VallÃľe Cedex 2, France.* `{gblin,vialette}@univ-mlv.fr`

Abstract

In this paper, we investigate the central problem of finding recombination events. It is commonly assumed that a present population is a descendent of a small number of specific sequences called *founders*. The recombination process consists in given two equal length sequences, generates a third sequence of the same length by concatenating the prefix of one sequence with the suffix of the other sequence. Due to recombination, a present sequence (called a *recombinant*) is thus composed of blocks from the founders. A major question related to founder sequences is the so-called MINIMUM MOSAIC problem: using the natural parsimony criterion for the number of recombinations, find the “best” founders. In this article, we prove that the MINIMUM MOSAIC problem given haplotype recombinants with no missing values is **NP**-hard when the number of founders is given as part of the input and propose some exact exponential-time algorithms for the problem, which can be considered polynomial provided some extra information. Notice that Rastas and Ukkonen proved that the MINIMUM MOSAIC problem is **NP**-hard using a somewhat unrealistic mutation cost function. The aim of this paper is to provide a better complexity insight of the problem.

1 Introduction

Given any two unrelated people, their DNA sequences differs by only about 0.1%. This small genetic variability is of particular importance since it influences how people differ with respect to risk of disease and response to drugs, and it represents 90% of the human genetic variation. A main challenge is to discover the DNA variants that contribute to common disease risk. These variations mostly arise on specific single positions called *Single Nucleotide Polymorphisms* (SNPs) where the corresponding nucleotides, called *alleles*, differ. To illustrate the SNPs, consider the three following DNA fragments from different individuals of a population:

$$\begin{array}{c} \ldots \mathrm{GGAC}\mathbf{C}\mathrm{TG}\ldots \\ \ldots \mathrm{GGAC}\mathbf{A}\mathrm{TG}\ldots \\ \ldots \mathrm{GGAC}\mathbf{T}\mathrm{TG}\ldots \end{array}$$

This SNP is composed of the three alleles in bold C, A and T. Gathering SNPs is very cheap when compared to the cost of full sequencing. Furthermore, each individual has two almost identical copies of all chromosomes except the sex chromosomes. Given a population, an *haplotype* refers to a combination of alleles at different positions on a chromosome. Therefore, each individual has two haplotypes (one maternal and one paternal) for each chromosome.

As a simple illustration, consider the following two partial haplotypes (a chromosome region where only the SNPs are shown)

$$\begin{array}{c} \ldots \mathrm{A} \ldots \mathrm{C} \ldots \mathrm{C} \ldots \mathrm{T} \ldots \mathrm{G} \ldots \mathrm{T} \ldots \\ \ldots \mathrm{A} \ldots \mathrm{C} \ldots \mathrm{A} \ldots \mathrm{G} \ldots \mathrm{C} \ldots \mathrm{T} \ldots \end{array}$$

The *genotype* corresponds to the set of alleles that a person has; thereby defining at each SNP the alleles of the two haplotypes. The corresponding genotype is

$$\ldots \mathrm{A/A} \ldots \mathrm{C/C} \ldots \mathrm{A/C} \ldots \mathrm{G/T} \ldots \mathrm{C/G} \ldots \mathrm{T/T} \ldots$$

Alleles of an SNP are called *heterozygous* if they differ (*e.g.* A/C), and *homozygous* otherwise (*e.g.* A/A). Since most SNPs are composed of only two alleles (among the 16 possibilities) that occur in a large percentage of the population, haplotypes are usually represented by binary sequences (one character for each of the two possible alleles per SNP). Genotypes, however, are usually represented by ternary sequences (0 and 1 denote the two homozygote alleles and 2 denotes the heterozygote one). Notice that it is common to denote missing values by the extra symbol "$-$". Therefore, the sequences are built over the alphabet $\{0, 1, 2, -\}$.

Genetic variation within species is mostly induced by a process called *recombination*. Given two equal length sequences, a recombination generates a third sequence of the same length by concatenating the prefix of one sequence with the suffix of the other sequence [1]. In the resulting sequence, the assembly point is referred to as a *breakpoint*. An illustration is given in Figure 1.

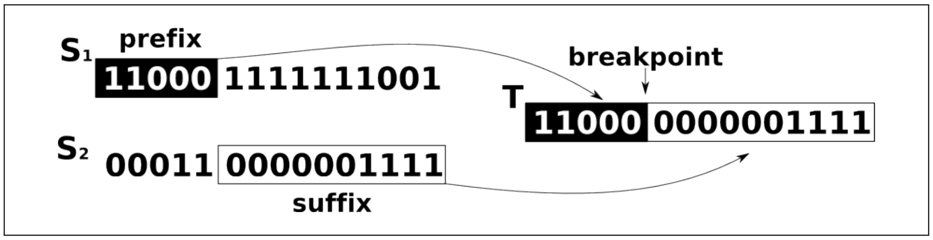


Figure 1: Recombination of S_1 and S_2 leading to T.

Finding recombination events has become a central problem in computational biology [2, 3, 4, 1, 5, 6]. In most combinatorial models, a present population is assumed to be a descendent of a small number of specific sequences called *founders*. Due to recombination, a present sequence, called a *recombinant*, is considered as composition of blocks from the founders. The term *mosaic* is often used to denote the mosaic-like structure of DNA induced by the recombinations [3] (cf. Figure 2). Indeed, considering one color assigned for each founder sequence, the coloration of the recombinants considering the colors of the founders induces a mosaic-alike pattern.

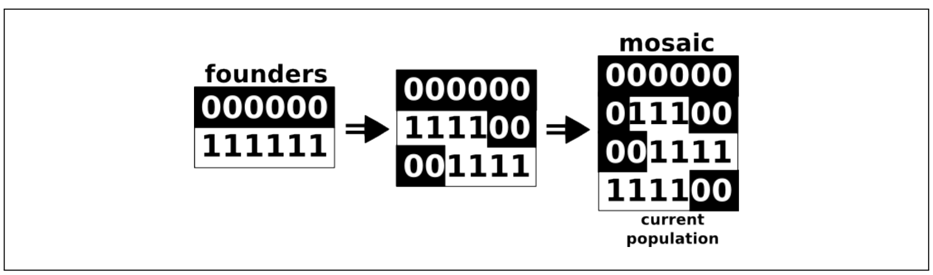


Figure 2: A Mosaic issued from a set of founders

A major question related to founder sequences is the so-called Minimum Mosaic problem [3]: using the natural parsimony criterion for the number of recombinations, find the "best" founders. More formally, the Minimum Mosaic problem is defined as follows: Given a set D of m equal n-length haplotype or genotype sequences (the current population) and a given number of founders K, find a set F of K founders inducing a minimum number of breakpoints.

There exists a dual problem, called the Minimum Segmentation problem [7]: given a set of founders F and a recombinant $r \in D$, the goal is to find a minimum number of segments of r (*i.e.* a segmentation), where each segment of r is inherited from the corresponding (*i.e.* same location) region of some founder.

This paper is organized as follows. In Section 2 we present the related state-of-the-art, and we introduce in Section 3 the needed material. Section 4 is devoted to computational complexity of the Minimum Mosaic problem. In

Section 5, we propose some exact exponential-time algorithms for this last.

2 Related works and known results

Ukkonen was the first to formulate an optimization problem based on the mosaic model and parsimony [3]. He considered two criteria: the number of founders and the number of recombinations in a solution. Ukkonen gave two polynomial-time algorithms: an $O(n(m + K^3))$ time algorithm for the MINIMUM SEGMENTATION problem and an $O(mn)$ time algorithm for the MINIMUM MOSAIC problem for $K = 2$ (Wu and Gusfield gave a similar result [6]). Also, Ukkonen designed an $O(nK^{O(m)})$ time dynamic programming algorithm for the general (*i.e.* $K \geq 2$) MINIMUM MOSAIC problem (the time complexity is not explicitly stated in the paper but will be given and justified in Section 5). According to the authors, this latter algorithm does not perform well for a moderate number of founders and/or recombinants.

A different but related problem was introduced by Gusfield [8] where there is no restriction on the number of founders. In the HAPLOTYPE INFERENCE problem, we are given a set of n genotype sequences and the goal is to find a set of n pairs of haplotype sequences (one pair per genotype) that is a good "explanation" of the given genotype sequences. For biological pertinence, a solution must be compatible with (or guided by) a given perfect phylogeny. Rastas and Ukkonen [5] observed that this latter problem is equivalent for $K > \sqrt{2m}$ to the MINIMUM MOSAIC problem for genotype recombinants without missing values (*i.e.*, $D \subset \{0, 1, 2\}^n$) – the latter being **NP**-complete [9].

In [5], Rastas and Ukkonen considered the MINIMUM MOSAIC problem with missing values and mutations (*i.e.* mismatches between founders and recombinants) leading to a different parsimony criterion. For each recombinant $r \in D$, a score $k + k'c$ is computed where (1) k is the number of breakpoints of a recombinant r' such that the Hamming distance between r and r' is k' (*i.e.* the number of mutations) and (2) c is the relative weight for a mutation compared to recombinations. Rastas and Ukkonen improved the complexity of MINIMUM SEGMENTATION problem by providing an $O(nmK)$ algorithm. They also proved that the MINIMUM MOSAIC problem for haplotype recombinants with possible missing values (*i.e.* $D \subset \{0, 1, -\}^n$) is **NP**-complete. Finally, they proved that even when missing values are forbidden, the MINIMUM MOSAIC problem for haplotype recombinants (*i.e.* $D \subset \{0, 1\}^n$) remains **NP**-complete for $K = 2$; it is equivalent to the **NP**-complete HYPERCUBE SEGMENTATION problem [10]. One should notice that this latter result is a bit "artificial", in the sense that the extra condition that the mutation cost $c = \frac{1}{nm}$ (necessary in the **NP**-hardness proof) roughly corresponds to forbid (and thus ignore) recombinations. Indeed, nm mutations become less expensive than a single recombination event. This is precisely the reason why Ukkonen [3] and Wu *et al.* [6] have been able to find a polynomial-time algorithm for the problem without mutation. Our contribution provide a better complexity insight of the problem.

A lower bound on the necessary breakpoints have been proposed by Wu *et* *al.* [11]. The main purpose of this last is to speed up application for finding exact minimum mosaic using branch and bound. Wu and Gusfield [6] designed an $O(nm)$ time algorithm for the MINIMUM MOSAIC problem for genotype recombinants without missing values (*i.e.* $D \subset \{0, 1, 2\}^n$) and $K = 2$. Furthermore, they gave an exact algorithm for the general case (haplotype recombinants without missing values). Notice that the time complexity of this latter algorithm is not given in the paper and the authors claim it to be practical for moderate n and m.

In [7], Zhang *et al.* investigated the MINIMUM SEGMENTATION problem for genotype recombinants and provided two dynamic programming algorithms with time complexity $O(nK^4)$ and $O(nK + PK^4)$, where P is the number of rows of the dynamic programming table. Recently, in [12], Roli *et al.* presented large neighbourhood search algorithms to tackle this problem. Any such algorithm performs a local search in which neighbourhoods have exponential size in the founder sequence length. Using an efficient tree-search technique, the neighbourhoods are exhaustively explored. Roli *et al.*, futhermore, evaluated these algorithms on three benchmark sets and finally, compared the best one to the state-of-the-art techniques.

3 Notations

In the rest of the paper, we do not consider missing values, *i.e.*, haplotypes are defined on $\{0, 1\}^n$. Given any string $s = s_1 s_2 \dots s_n$, and two integers i and j, $1 \leq i \leq j \leq n$, we denote by $s[i, j]$ the substring $s_i s_{i+1} \dots s_j$ of s. Given a set of haplotype founders $f_1, f_2, \dots, f_K$, each in $\{0, 1\}^n$, and an haplotype recombinant r of size n, a *segmentation* of r from $f_1, f_2, \dots, f_K$ is a partition of the interval $[1 \dots n] = \{1, 2, \dots, n\}$ into consecutive intervals $I_1, I_2, \dots, I_k$ such that, for each $1 \leq i \leq k$, we have $r[I_i] = f_j[I_i]$ for some $j \in \{1, 2, \dots, K\}$. The *cost* of the segmentation is $k - 1$. For example, for $r = 001011$, $f_1 = 000000$ and $f_2 = 111111$, $\{I_1 = f_1[1 \dots 2], I_2 = f_2[3 \dots 3], I_3 = f_1[4 \dots 4], I_4 = f_2[5 \dots 6]\}$ is a segmentation of r out from f_1 and f_2 of cost 3. The cost of segmenting r out from $f_1, f_2, \dots, f_K$ is denoted $\text{cost}(r; f_1, f_2, \dots, f_K)$ (the cost of a best segmentation can be found in $O(nK)$ time [5]).

The MINIMUM MOSAIC problem can be defined as follows:

MINIMUM MOSAIC

- **Input** : A set of m recombinants $D = \{r_1, r_2, \dots, r_m\}$, $r_i \in \{0, 1\}^n$ for $1 \leq i \leq m$, and an integer K
- **Solution** : A set of K founders $F = \{f_1, f_2, \dots, f_K\}$, $f_j \in \{0, 1\}^n$ for $1 \leq j \leq K$, such that $\sum_{i=1}^{m} \text{cost}(r_i; f_1, f_2, \dots, f_K)$ is minimized.

4 Hardness result for the Minimum Mosaic problem

In this section, we prove the Minimum Mosaic problem when K is part of the input to be **NP**-hard without relying on unrealistic prohibitive mutation costs (as done in [5]). For the sake of presentation, we first generalize the problem to arbitrary strings. The following lemma proves that this can be done safely.

Lemma 1 *If the* Minimum Mosaic *problem on arbitrary strings is* **NP***-hard, then so is the* Minimum Mosaic *problem on binary strings.*

Proof: Assume to be given a natural K and a set of m recombinants $D = \{r_1, r_2 \dots r_m\} \subset \Sigma^n$ where $\Sigma = \{\sigma_1, \sigma_2 \dots \sigma_k\}$ is any alphabet on k symbols. Then, take any encoding of the symbols in Σ by binary strings of length $\lceil \log_2 k \rceil$. In other words, let $\delta : \Sigma \mapsto \{0,1\}^{k'}$ be any injection from Σ to $\{0,1\}^{k'}$ with $k' = \lceil \log_2 k \rceil$. Once such an encoding has been fixed, we can extend δ to get an injective function which maps every string r_i over Σ into a binary string $\delta(r_i)$ of length $|\delta(r_i)| = k' \, |r_i|$. Notice that any feasible solution $\langle f_1, f_2 \dots f_K \rangle$ for the instance $\langle K, r_1, r_2 \dots r_m \rangle$ naturally maps into the feasible solution $\langle \delta(f_1), \delta(f_2), \dots, \delta(f_K) \rangle$ for the instance $\langle K, \delta(r_1), \delta(r_2) \dots \delta(r_m) \rangle$, and the cost remains unaffected.

Let us show that the converse is also true. To do so, we claim that, given any feasible solution $\langle f'_1, f'_2 \dots f'_K \rangle$ for the instance $\langle K, \delta(r_1), \delta(r_2) \dots \delta(r_n) \rangle$, we can always modify it, without increasing its cost, in such a way that each of the f'_i is actually the binary encoding of some string over Σ. In other words, we can assume that $f'_1 = \delta(f_1), f'_2 = \delta(f_2) \dots f'_K = \delta(f_K)$. In this way, the converse would directly follow. In order to prove our claim, we can actually act on the f'_i's and "clean" them out one by one. Assume f'_i is not cleaned, that is, f'_i does not belong to the image of map δ. Then there exists some $1 \leq j \leq n$ such that $f'_i[k'(j-1)+1, k'j]$ does not belong to the codewords set $\Delta(\Sigma) := \{\delta(\sigma) \ : \ \sigma \in \Sigma\}$. In this case, where $\overline{\sigma}$ is any symbol in Σ such that $\delta(\overline{\sigma})$ has a longest suffix in common with $f'_i[k'(j-1)+1, k'j]$, we modify f'_i precisely on the interval $[k'(j-1)+1, k'j]$, and, more precisely, by replacing it by the string $\delta(\overline{\sigma})$. Considering the way the procedure for finding a minimum cost segmentation operates, it is easy to see that this modification does not increase the cost. Indeed, given the set of founders, finding the segmentation of a given row r consists in iteratively "cover" the position of r with longest common substrings (*e.g.* Algorithm 1). □

In order to facilitate the understanding of the proof, let us further strengthen the formulation of our problem: consider the more general formulation where K' of the K founders comprising the solution (and to be given in output) are actually given as specified in advance in the input. We will refer to those specific founders as *forced founders* and will first show how one can force a part of the founders.

Algorithm 1 Algorithm for finding a minimum cost segmentation : $FS(r; f_1, f_2, ..., f_K)$

Let $i = |r|$
if $i = 0$ **then**
  return
end if
Let j be the smallest positive natural such that $r[j..i] = f_k[j..i]$ for some k
if $j > i$ **then**
  return "No production exists"
end if
return $FS(r[1, j-1]; f_1[1, j-1], f_2[1, j-1], \ldots, f_K[1, j-1]) + [j..i]$

Indeed, here is a polynomial-time reduction from any instance containing forced founders to an instance without ones: for each forced founder f^f, add $n\,m$ copies of f^f in the recombinants set D. It is clear that in a solution, one has to include f^f in the founder set, otherwise, it will induce at least $n\,m$ breakpoints which is the maximal number of breakpoints one can get in the original instance. Note that given an objective of at most B breakpoints, it suffices to add B copies of f^f to get this property.

Thus, for the purpose of the reduction, we can without loss of generality assume to have instances of the form $\langle K, D, F^f \rangle$ such that $D = \{r_1, r_2 \ldots r_{m''}\} \subset \Sigma^n$ is the set of recombinants and $F^f = \{f_1^f, f_2^f \ldots f_{m'}^f\} \subset \Sigma^n$ is the set of forced founders. Let us now provide a reduction from the **NP**-hard VERTEX-COVER problem:

VERTEX-COVER

• **Input** : A graph $G = (V, E)$ such that $|V| = n_G$, $|E| = m_G$ and an integer k_G

• **Solution** : A subset V' of V such that each edge of G is incident to at least one vertex of V' and $|V'| \leq k_G$

Our reduction begins by giving an arbitrary orientation to each edge of G, that is, for each $e_j \in E$, let t_j and h_j be indices such that v_{t_j} is the tail and v_{h_j} is the head of the arbitrary oriented edge e_j.

We consider the alphabet $\Sigma := \{W, Z\} \cup \{X_i \ : \ i = 1, 2, \ldots n_G\}$. Informally, there is one letter for each vertex of V, while W and Z act like separators letters. We define $m = C\,n_G + 3$ recombinants, each of length $n = 6\,m_G$ built as follows (C is a constant defined later):

- $r_1 = (WWZZWW)^{m_G}$,
- $r_2 = \prod_{j=1}^{m_G}(ZZX_{t_j}X_{t_j}ZZ)$ and $r_3 = \prod_{j=1}^{m_G}(ZZX_{h_j}X_{h_j}ZZ)$,
- $r_i^j = (X_iX_iX_iX_iX_iX_i)^{m_G}$, for each $1 \leq i \leq n_G$, $1 \leq j \leq C$.

We then define a set F^f of $K' = 1 + 2m_G n_G$ forced founders as follows.

- $F_1^f = (ZZZZZZ)^{m_G}$,
- $F_{i,t}^f = Z^{3\,t} X_i X_i X_i Z^{3\,(2m_G - t - 1)}$, for each $0 \leq t \leq 2m_G - 1$, and $1 \leq i \leq n_G$.

Finally, we set $K = 2m_G n_G + k_G + 2$; that is asking for $K'' = k_G + 1$ founders. An example of this construction is shown Figure 3. The idea is that k_G of these founders correspond to the k_G vertices in the vertex cover while the single extra founder will collect, from each one of the edges, the endpoint (at most one) not included in the vertex cover.

Lemma 2 *If the graph $G = (V, E)$ admits a vertex cover of size at most k_G then the corresponding built instance of our problem admits a solution of cost at most $C(n_G - k_G)(2m_G - 1) + 6m_G$.*

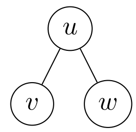
Proof: Let $V' \subseteq V$ be a vertex cover of size k_G of G. For each $1 \leq j \leq m_G$, we let c_j be an index such that $v_{c_j} \in V' \cap e_j$ (such an index exists since V' is a vertex cover). By definition, $c_j \in \{t_j, h_j\}$. Let $\overline{c_j} = \{t_j, h_j\} \backslash c_j$. We construct our solution with the following set F' of $K'' = k_G + 1$ founders:

- $F'_0 = \prod_{j=1}^{m_G} (WWX_{\overline{c}_j} X_{\overline{c_j}} WW)$,
- $F'_i = (X_i X_i X_i X_i X_i X_i)^{m_G}$, for each $v_i \in V'$.

Let us now compute the corresponding cost. First, notice that regarding r_1, the associated cost is $cost(r_1; F' \cup F^f) = 2\,m_G$. Indeed, we have to switch from F'_0 to F_1^f and from F_1^f to F'_0 for each m_G blocks of length 6. Then, considering r_2 and r_3, the corresponding costs are $cost(r_2; F' \cup F^f) = 2\,m_G$ and $cost(r_3; F' \cup F^f) = 2\,m_G$. Indeed, for each $1 \leq j \leq m_G$, since V' is a vertex cover, F'_{c_j} and F'_0 will prevent a breakpoint between the X_{t_j}'s (and between the X_{h_j}'s) since $c_j \in \{t_j, h_j\}$ and $\overline{c_j} = \{t_j, h_j\} \backslash c_j$. Therefore, by switching from F_1^f to F'_{c_j} or F'_0 and back to F_1^f for each m_G blocks of length 6, it only induces 2 breakpoints for each block. Finally, for $1 \leq j \leq C$, $cost(r_i^j; F' \cup F^f) = 0$ for each $v_i \in V'$ whereas $cost(r_{i'}^j; F' \cup F^f) = C(2\,m_G - 1)$ for each $v_{i'} \notin V'$. Indeed, considering each recombinants $r_{i'}^j$ such that $v_{i'} \notin V'$, by switching from $F_{i',2t-2}^f$ to $F_{i',2t-1}^f$ for the t^{th} block of length 6, it will only induce 2 breakpoints for each block (except the last one). Since $|V'| = k_G$, there are $n_G - k_G$ recombinants which costs $(2m_G - 1)$ each.

On the whole, $\sum_{i=1}^{2} cost(r_i; F' \cup F^f) + \sum_{i=1}^{n_G} \sum_{j=1}^{C} cost(r_i^j; F' \cup F^f) = 2m_G + (2 \times 2m_G) + C(n_G - k_G)(2m_G - 1) = C(n_G - k_G)(2\,m_G - 1) + 6\,m_G$. $\square$

We now turn to considering the reverse direction. As a technical hint, two main driving forces here are that the extra founder will essentially be enforced by feasibility (each recombinant must be obtained from the founders) whereas the k_G covering founders will be enforced by parsimony, assuming C is sufficiently big.



G

r_1 :	W	W	Z	Z	W	W	W	W	Z	Z	W	W	
r_2 :	Z	Z	X_u	X_u	Z	Z	Z	Z	X_u	X_u	Z	Z	
r_3 :	Z	Z	X_v	X_v	Z	Z	Z	Z	X_w	X_w	Z	Z	
r_u^j :	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u	C times
r_v^j :	X_v	X_v	X_v	X_v	X_v	X_v	X_v	X_v	X_v	X_v	X_v	X_v	C times
r_w^j :	X_w	X_w	X_w	X_w	X_w	X_w	X_w	X_w	X_w	X_w	X_w	X_w	C times

F_1^f :	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
$F_{u,0}^f$:	X_u	X_u	X_u	Z	Z	Z	Z	Z	Z	Z	Z	Z
$F_{u,1}^f$:	Z	Z	Z	X_u	X_u	X_u	Z	Z	Z	Z	Z	Z
$F_{u,2}^f$:	Z	Z	Z	Z	Z	Z	X_u	X_u	X_u	Z	Z	Z
$F_{u,3}^f$:	Z	Z	Z	Z	Z	Z	Z	Z	Z	X_u	X_u	X_u
$F_{v,0}^f$:	X_v	X_v	X_v	Z	Z	Z	Z	Z	Z	Z	Z	Z
$F_{v,1}^f$:	Z	Z	Z	X_v	X_v	X_v	Z	Z	Z	Z	Z	Z
$F_{v,2}^f$:	Z	Z	Z	Z	Z	Z	X_v	X_v	X_v	Z	Z	Z
$F_{v,3}^f$:	Z	Z	Z	Z	Z	Z	Z	Z	Z	X_v	X_v	X_v
$F_{w,0}^f$:	X_w	X_w	X_w	Z	Z	Z	Z	Z	Z	Z	Z	Z
$F_{w,1}^f$:	Z	Z	Z	X_w	X_w	X_w	Z	Z	Z	Z	Z	Z
$F_{w,2}^f$:	Z	Z	Z	Z	Z	Z	X_w	X_w	X_w	Z	Z	Z
$F_{w,3}^f$:	Z	Z	Z	Z	Z	Z	Z	Z	Z	X_w	X_w	X_w

$F_0^{'}$:	W	W	X_v	X_v	W	W	W	W	X_w	X_w	W	W
$F_1^{'}$:	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u	X_u

Figure 3: An example of our construction, from a graph G with $n_G = 3$ nodes and $m_G = 2$ edges. We also set k_G to 1. From this graph, $n_G \times C + 3 = 3C + 3$ recombinants are built, and $1 + 2 \times m_G \times n_G = 13$ forced founders sequences are set – hence only $k_G + 1 = 2$ founders have to be found. They are shown as $F_0^{'}$ and $F_1^{'}$, and the associated breakpoints in the recombinants are drawn (there is a breakpoint between one gray block and one white block, and a breakpoint between one white block and one gray block).

Lemma 3 *Given a graph $G = (V, E)$ and the corresponding built instance of our problem, if the latter one admits a solution of cost at most $C(n_G - k_G)(2m_G - 1) + 6m_G$, and $C > 6m_G$, then G admits a vertex cover of size at most k_G.*

Proof: Let us prove that the cost of recombinants r_i^j, $1 \leq i \leq n_G$, $1 \leq j \leq C$, for any solution is greater than $C((n_G - k_G)(2m_G - 1))$. Note first that, only considering the set of forced founders F^f, each r_i^j, $1 \leq i \leq n_G$, $1 \leq j \leq C$, has a $cost(r_i^j, F^f) = C(n_G(2m_G - 1))$. Indeed, given a recombinant r_i^j, one has to switch from $F_{i,t}^f$ to $F_{i,t+1}^f$ for $0 \leq t \leq 2m_G - 2$.

Considering now both F^f and F, if the set of $K'' = k_G + 1$ founders is composed of exact copies of K'' different recombinants $\{r_{i_1}^1, r_{i_2}^1 \dots r_{i_{k_G+1}}^1\}$ (for some $1 \leq i_1, i_2 \dots i_{k_G+1} \leq n_G$), then $\sum_{i=1}^{n_G} \sum_{j=1}^{C} cost(r_i^j; F' \cup F^f) = C(n_G - (k_G + 1))(2m_G - 1)$. However, r_1 is built over some W letters; which does not belong to any forced founders. Therefore, there should be founders among the K'' including a W in positions $6t+1, 6t+2, 6t+5, 6t+6$, for each $0 \leq t \leq m_G - 1$. Then, for any $0 \leq t \leq m_G - 1$, there will exist among the recombinants $\{r_{i_1}^1, r_{i_2}^1 \dots r_{i_{k+1}}^1\}$ one recombinant – say $r_{i_1}^1$ – that will induce a switch (for each of its C copies) from a founder of F to $F_{i_1,t+1}^f$ (resp. $F_{i_1,t+2}^f$) due to the WW at positions $6t+1, 6t+2$ (resp. $6t+5, 6t+6$) in the corresponding founder. Therefore, on the whole, one will end-up with an extra cost of $2Cm_G$. In the best case, one should only "sacrifice" one of the K'' founders (which was only allowing a gain of $2C(m_G - 1)$ breakpoints). On the whole, $\sum_{i=1}^{n_G} \sum_{j=1}^{C} cost(r_i^j; F' \cup F^f) \geq C(n_G - k_G)(2m_G - 1)$. Moreover, considering the objective cost (*i.e.* $C(n_G - k_G)(2m_G - 1) + 6m_G$) and that $C > 6m_G$, the following is enforced.

Property 4 *Considering any position, each of the K'' founders has a different letter.*

Proof: By contradiction, suppose there are two founders with the same letter in a given position. Then, there is at least one more breakpoint for each copy of a given recombinant of $\{r_{i_1}^{j_1}, r_{i_2}^{j_2} \dots r_{i_{k_G+1}}^{j_{k_G+1}}\}$ which leads to a cost above the objective. □

Property 5 *A founder with a letter X_u in a given position contains only letters X_u in the next positions until a letter W or the end of the founder is encountered.*

Proof: By contradiction, in an optimal solution, suppose there is a founder with a letter X_u at position k followed by a letter X_v. Then, due to Property 4, there should not exist another founder such that X_u is at position k nor X_v at position $k+1$. Therefore, there is a breakpoint between positions k and $k+1$ in all r_u^j (resp. r_v^j); that is $2C$ breakpoints. Consider now changing X_v into X_u in the corresponding founder. Then it will induce at least C less breakpoints, leading to a better solution; a contradiction. □

Property 6 *All the W letters occur in the same founder.*

Proof: Let us first prove that W letters at position $6t+1$ and $6t+2$ belong to the same founder. By contradiction, suppose it is not the case; *e.g.* there is a W letter at position $6t+1$ in founder F_i and a W letter at position $6t+2$ in founder F_j, $i \neq j$. By Property 4, there is no W letter at position $6t+2$ (resp. $6t+1$) in founder F_i (resp. F_j). Thus, there is a breakpoint in the recombinant r_1 between positions $6t+1$ and $6t+2$. Consider now swapping $F_i[1..6t+1]$ and $F_j[1..6t+1]$. This will decrease the number of breakpoints at least by one since it only changes the cost due to positions $6t+1$ and $6t+2$. Same argument holds for the other W positions.

We now know that the W letters appears by block of size 4 (except for the first and last blocks; which are of size 2). It also appears that if the blocks of letters W are not on the same founder, it creates at least C more breakpoints. Figure 4 illustrates this configuration. Indeed, in an optimal solution, consider two founders F_i and F_j such that there are W letters at positions $6t+1$ and $6t+2$ (resp. $6t+5$ and $6t+6$) in F_i (resp. F_j). Assume, moreover, that a X_u (resp. X_v) appears in position $6t+3$ (resp. $6t+4$) in F_i (resp. F_j). Then considering the forced founders, F_i and F_j, there will be one breakpoint between positions $6t+3$ and $6t+4$ in all r_u^x and r_v^x, for $1 \leq x \leq C$ and one breakpoint between positions $6t+6$ and $6t+7$ in all r_v^x, for $1 \leq x \leq C$. Consider now swapping $F_i[1..6t+4]$ and $F_j[1..6t+4]$ and replacing the X_i's of F_j by X_j's. It will induce at least C less breakpoints, leading to a better solution; a contradiction. $\square$

In the following, let us assume that F_0 denotes the founder containing the W letters; the other founders are referred to as F_1, F_2,..., $F_{K''}$. Now that we know the precise topology of the founders of any optimal solution, we can now prove that $\{F_1, F_2 \ldots F_{K''}\}$ corresponds to a vertex cover in G. Let $V' \subseteq V$ be defined as follows. For each founder F_i, $1 \leq i \leq K''$, if F_i is the exact copy of the recombinant r_x^y, then add v_x in V'. Let us prove that V' is a vertex cover of G. By contradiction, suppose it is not. Then, it exists at least one y such that $(v_{h_y}, v_{t_y}) \in E$ and both $\{v_{h_y}, v_{t_y}\} \cap V' = \emptyset$. Therefore, there is at least a breakpoint between X_{t_y} and X_{t_y} in r_2 or between X_{h_y} and X_{h_y} in r_3 since neither h_y nor t_y appears in $\{F_1, F_2 \ldots F_{K''}\}$, and F_0 can contain at most one of the two in the corresponding positions. It is worth noting that none of the forced founders can prevent these breakpoints since they never have the same letters at position $6t+3$ and $6t+4$. $\square$

With Lemma 2 and Lemma 3, we obtain the following result.

Proposition 7 *The* Minimum Mosaic *problem given haplotype recombinants with no missing values is* **NP***-hard.*

W not on the same founder

a) $...(X_u X_u X_u X_u X_u X_u)(X_u ...$
$...(X_v X_v X_v X_v X_v X_v)(X_v ...$

b) $...(W\ W\ X_u X_u X_u X_u)(X_u ...$
$...(X_v X_v X_v X_v W\ W\ )(W\ ...$

c) $...(X_u X_u X_u Z\ \ Z\ \ Z\ \ )(Z\ \ ...$
$...(Z\ \ Z\ \ Z\ \ X_u X_u X_u)(Z\ \ ...$
$...(X_v X_v X_v Z\ \ Z\ \ Z\ \ )(Z\ \ ...$
$...(Z\ \ Z\ \ Z\ \ X_v X_v X_v)(Z\ \ ...$

W on the same founder

a) $...(X_u X_u X_u X_u X_u X_u)(X_u ...$
$...(X_v X_v X_v X_v X_v X_v)(X_v ...$

b) $...(W\ W\ X_u X_u W\ W\ )(W\ ...$
$...(X_v X_v X_v X_v X_v X_v)(X_v ...$

c) $...(X_u X_u X_u Z\ \ Z\ \ Z\ \ )(Z\ \ ...$
$...(Z\ \ Z\ \ Z\ \ X_u X_u X_u)(Z\ \ ...$
$...(X_v X_v X_v Z\ \ Z\ \ Z\ \ )(Z\ \ ...$
$...(Z\ \ Z\ \ Z\ \ X_v X_v X_v)(Z\ \ ...$

Figure 4: a) The recombinants r_u^x and r_v^y. b) Two founders involving X_u and X_v. c) Forced founders. The created breakpoints in the recombinants induced by b) and c) are between gray and white blocks.

5 Exact algorithms for the Minimum Mosaic problem

In this section, we will provide exact algorithms considering a variant of the Minimum Mosaic/Segmentation problems where extra information on the breakpoints are given (*e.g.* position, number, ...).

5.1 An FPT algorithm for the Minimum Mosaic problem

Let us first give a complexity study of the general solution proposed by Ukkonen in Section 4 of [3]. Indeed, the corresponding solution shows that, once the number of founders (*i.e.* K) is a constant, then the minimum mosaic problem is in FPT. The main idea of the dynamic programming solution of Ukkonen is to compute every partition of size K of the m input recombinants for each column. The central recurrence is based on the fact that the "best" K-partitions for the i first columns can be computed using one of all the "best" K-partitions of the $i-1$ first columns and any K-partition of the i^{th} column; the parsimony criterion being the number of breakpoints induced. In a dynamic programming table, one store the minimum cost of any K-partition of the i first columns for all the K-partitions of the i^{th} column. On the whole, since (1) there are K^m K-partitions of any column i and (2) one has to compute the minimum number of breakpoints considering all the K-partitions of column $i-1$ for each K-partition of column i, the time complexity is $O(nK^{2m})$; which leads to a polynomial solution when the number of recombinants m is a constant. It also provides a practical solution when the number of founders K is a constant.

5.2 An algorithm when the location of the breakpoints is known

Let us now design exact algorithms for a variant of the MINIMUM MOSAIC/SEGMENTATION problems that considers that extra informations on the breakpoints are given. Let us first consider a kind of reverse problem of MINIMUM SEGMENTATION problem where given a set of m recombinants $D = \{r_1, r_2, \ldots, r_m\} \subset \{0,1\}^n$ and a set of B identified breakpoints on D – *i.e.* an overall cost B segmentation of the recombinants $S = \{I_1^1, I_2^1 \ldots I_{k_1}^1, \ldots I_1^m, I_2^m \ldots I_{k_m}^m\}$ such that $(\sum_{i=1}^m k_i) - m = B$ and I_y^x is the y^{th} segment of recombinant r_x – find a set of K founders $F = \{f_1, f_2, \ldots, f_K\} \subset \{0,1\}^n$ such that the B cost segmentation can be derived from F. We propose a polynomial-time algorithm (Algorithm 2) that solves this problem. For any I_y^x, let L_y^x (resp. R_y^x) denote the leftmost (resp. rightmost) position of I_y^x in $[1..n]$.

Let us now prove that this algorithm indeed find a solution if one exists. Roughly, the algorithm tries to reassemble the segments in order to produce at most K founders. To do so, the algorithm computes the founders from left to right using the available segments in L. For each available segment I_y^x, the algorithm tries first to detect if there is a founder that has the longest common prefix with I_y^x at the given position L_y^x. If there exist one then the corresponding founder is "merged" with I_y^x for the positions from L_y^x to R_y^x. Otherwise, the algorithm tries to find a founder such that the position L_y^x is empty and then merge the founder and I_y^x.

Let us prove that, if there exist a solution then the algorithm will find it. First, notice that if there exist such a solution then any segments can be placed entirely in one of the K founders. Thus when considering all the segments starting at a given position, one should be able to find a distribution of the segments among the K founders. This ensures that when considering I_y^x, the choice of the founder for I_y^x will not interfere with placement of $I_y'^{x'}$. Indeed, either in the solution segments I_y^x and $I_y'^{x'}$ originated from the same founder and thus will have a common prefix or segments I_y^x and $I_y'^{x'}$ were obtained from

Algorithm 2 Find K founders according to a segmentation S

Let L be a sorted list – according to the leftmost position – of elements of S
$F = \{F_1, F_2 \ldots F_K\}$ such that $F_i = [\text{""}, \text{""}, \ldots, \text{""}]$, $1 \le i \le K$
while L is not empty **do**
  $I_y^x \leftarrow pop(L)$ //remove and return head element of L
  $LCP = 0$, $F' = null$
  for each F_i in F **do**
    $Z \leftarrow$ the leftmost empty position in F_i greater than L_y^x
    if $F_i[L_y^x..Z]$ and I_y^x share a common prefix **then**
      if $LCP < Z - L_y^x$ **then**
        $F' = F_i$
        $LCP = Z - L_y^x$
      end if
    end if
  end for
  if $F'! = null$ **then**
    $F'[L_y^x, R_y^x] \leftarrow I_y^x$
    goto $\_end$;
  end if
  //No common prefix found
  //Find the first empty position
  for each F_i in F **do**
    if $F_i[L_y^x]$ is empty **then**
      $F_i[L_y^x, R_y^x] \leftarrow I_y^x$
      goto $\_end$;
    end if
  end for
  Exit with error
  //This case is only reachable
  //if no solution can be found
  $\_end:$
end while
return F

different founders. This leads to an overall $O(BKL)$ time-complexity algorithm where L is the length of the longest segment.

5.3 An algorithm when the number of breakpoints for each recombinant is known

Let us now consider the case where only the number of breakpoints for each recombinant is known (*i.e.* without the positions of these breakpoints). Then, one can test all the possible positions for each recombinant and apply the previously proposed algorithm. On the whole, since given a recombinant r_i of length n with b_i breakpoints, there are at most $\binom{n}{b_i} = O(n^{b_i})$ possibilities of placement of the breakpoints, the corresponding algorithm runs in $O(n^{b_1}.n^{b_2}.\ldots.n^{b_m}.BKL) = O(n^{b_1+b_2+\cdots+b_m}.BKL) = O(n^B.BKL)$.

5.4 An algorithm when the number of breakpoints is known

Finally, we give an algorithm with a lower complexity than the algorithm in the previous section in the case one only knows the number B of allowed breakpoints. Let first give the maximum number of different strings in D, that is the number m_d of different recombinants among the m recombinants of D. In the worst case, there is only one breakpoint on each recombinant, that is B different recombinants. Moreover, there is a maximum number of K different recombinants with no breakpoint in D (they are copies of the founders). Thus, we can give the following upperbound for the number of different recombinants in D : $m_d \leq K + B$. We also have $K \leq m_d \leq m$.

Let now decide which recombinants in D will have some breakpoints. Therefore, we have to choose B recombinants among the m_d recombinants with at least one breakpoint. We will try all different possibilities. There are $\binom{m_d}{B} = \binom{K+B}{B} = (K + B)^B$ different configurations. For each one, run the exact $O(nK^{2m})$ algorithm of Ukkonen to find the best possible founders. This algorithm is ran among a set of only B chosen recombinants. Thus, the running time of the exact algorithm is $O(nK^{2B})$. On the whole, the total running time of the algorithm when only the number of breakpoints is known is $O((K + B)^B \times nK^{2B})$. This last result demonstrates that once the number of breakpoints is a constant the problem becomes polynomial.

6 Open problems

In this article, we showed that the Minimum Mosaic problem given haplotypes with no missing values is hard when the number K of founders is given as part of the input. When K is a constant but the number m of recombinants is not, the problem is still widely open. Indeed, there is an ocean between the linear complexity of the problem when m (and thus K) is a constant, the polynomial-time complexity when $K = 2$, and the **NP**-hardness when K is unbounded. It is also widely open whether the problem admits some PTAS since our reduction does not preserve the approximation features.

7 Acknowledgement

The authors acknowledge partial funding from DFG PABI BO1910/9-1 and ANR project BIRDS JCJC SIMI 2-2010, and also would like to thanks the anonymous reviewers for valuable comments and remarks.

References

[1] Mikko Koivisto, Pasi Rastas, and Esko Ukkonen. Recombination systems. In Juhani Karhumäki, Hermann A. Maurer, Gheorghe Paun, and Grzegorz Rozenberg, editors, *Theory Is Forever*, volume 3113 of *Lecture Notes in Computer Science*, pages 159–169. Springer, 2004.

[2] John D. Kececioglu and Dan Gusfield. Reconstructing a history of recombinations from a set of sequences. *Discrete Applied Mathematics*, 88(1-3):239–260, 1998.

[3] Esko Ukkonen. Finding founder sequences from a set of recombinants. In Roderic Guigó and Dan Gusfield, editors, *Algorithms in Bioinformatics, Second International Workshop (WABI)*, volume 2452 of *Lecture Notes in Computer Science*, pages 277–286. Springer, 2002.

[4] Russell Schwartz, Andrew G. Clark, and Sorin Istrail. Inferring piecewise ancestral history from haploid sequences. In Sorin Istrail, Michael S. Waterman, and Andrew G. Clark, editors, *Computational Methods for SNPs and Haplotype Inference*, volume 2983 of *Lecture Notes in Computer Science*, pages 62–73. Springer, 2002.

[5] Pasi Rastas and Esko Ukkonen. Haplotype inference via hierarchical genotype parsing. In Raffaele Giancarlo and Sridhar Hannenhalli, editors, *Algorithms in Bioinformatics, 7th International Workshop (WABI)*, volume 4645 of *Lecture Notes in Computer Science*, pages 85–97. Springer, 2007.

[6] Yufeng Wu and Dan Gusfield. Improved algorithms for inferring the minimum mosaic of a set of recombinants. In Bin Ma and Kaizhong Zhang, editors, *Combinatorial Pattern Matching, 18th Annual Symposium (CPM)*, volume 4580 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2007.

[7] Qi Zhang, Wei Wang, Leonard McMillan, Jan Prins, Fernando Pardo-Manuel de Villena, and David Threadgill. Genotype sequence segmentation: Handling constraints and noise. In Keith A. Crandall and Jens

Lagergren, editors, *Algorithms in Bioinformatics, 8th International Workshop (WABI)*, volume 5251 of *Lecture Notes in Computer Science*, pages 271–283. Springer, 2008.

[8] Dan Gusfield. Haplotyping as perfect phylogeny: conceptual framework and efficient solutions. In *6th International Conference on Computational Biology (RECOMB)*, pages 166–175, 2002.

[9] Giuseppe Lancia, Maria Cristina Pinotti, and Romeo Rizzi. Haplotyping populations by pure parsimony: Complexity of exact and approximation algorithms. *INFORMS Journal on Computing*, 16(4):348–359, 2004.

[10] Jon M. Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan. Segmentation problems. In *13th ACM Symp. Theory of Computing (STOC)*, pages 473–482, 1998.

[11] Yufeng Wu. Bounds on the minimum mosaic of population sequences under recombination. In Amihood Amir and Laxmi Parida, editors, *Combinatorial Pattern Matching, 21st Annual Symposium (CPM)*, volume 6129 of *Lecture Notes in Computer Science*, pages 152–163. Springer, 2010.

[12] Andrea Roli, Stefano Benedettini, Thomas Stützle, and Christian Blum. Large neighbourhood search algorithms for the founder sequence reconstruction problem. *Computers & OR*, 39(2):213–224, 2012.