Minimum Mosaic Inference of a Set of Recombinants
Guillaume Blin, Romeo Rizzi, Florian Sikora, Stéphane Vialette

To cite this version:

HAL Id: hal-00620371
https://hal-upec-upem.archives-ouvertes.fr/hal-00620371
Submitted on 22 Oct 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract
In this paper, we investigate the central problem of finding recombination events (Kececioglu & Gusfield 1998, Ukkonen 2002, Schwartz et al. 2002, Koivisto et al. 2004, Rastas & Ukkonen 2007, Wu & Gusfield 2007). It is commonly assumed that a present population is a descendent of a small number of specific sequences called founders. 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 hard for an unbounded number of founders and propose some exact exponential-time algorithms for the problem. Notice that, in (Rastas & Ukkonen 2007), Rastas et al. proved that the Minimum Mosaic problem is hard using a somewhat unrealistic mutation cost function (details provided afterwards). The aim of this paper is to provide a better complexity insight of the problem.

Keywords: Minimum Mosaic problem, SNP, Haplotype

1 Introduction
Given any two unrelated people, their DNA sequences will only differ on about 0.1%. This small genetic variability is of particular importance since it influences how people differ in their risk of disease or their response to drugs. A main challenge is to discover the DNA variants that contribute to common diseases how people differ in their risk of disease or their response to drugs. A main challenge is to discover the DNA variants that contribute to common diseases.

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) whereas genotypes 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 on 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 (Koivisto et al. 2004). In the resulting sequence, the assembly point is referred as a breakpoint. An illustration is given in Figure 1.

![Figure 1: Recombination of $S_1$ and $S_2$ leading to $T$.](image)

Finding recombination events has become a central problem in computational biology (Kececioglu & Gusfield 1998, Ukkonen 2002, Schwartz et al. 2002, Koivisto et al. 2004, Rastas & Ukkonen 2007, Wu & Gusfield 2007). 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 thus composed of blocks from the founders. The term mosaic is often used to denote the mosaic-like structure of DNA induced by the recombinations (Ukkonen 2002) (c.f. Figure 2).

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. More formally, the Minimum Mosaic problem, in its natural decisional form, is defined as follows: Given a set $D$ of $m$ recombining sequences, find the set $F$ of $k$ founders such that the number of recombinations needed to obtain the sequences from the founders is minimized.

The corresponding genotype is

$$\ldots A/A \ldots C/C \ldots G/T \ldots C/G \ldots T/T \ldots$$

$$\ldots A/A \ldots C/C \ldots A/C \ldots G/T \ldots C/G \ldots T/T \ldots$$

the SNPs are shown)
equal n-length haplotype or genotype sequences (the current population) and a given number of founders K, find a set F of K founders that induce a minimum number of breakpoints. There exists a dual problem, called the Minimum Segmentation problem, where we are given a set of founders F and a recombinant r ∈ D and the goal is to find a minimal segmentation of r, 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. In Section 5, we propose some exact exponential-time algorithms for the Minimum Mosaic problem.

2 Related works and known results

Ukkonen first formulated an optimization form of the problem based on the mosaic model and parsimony (Ukkonen 2002). 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(nmK) time algorithm for the Minimum Mosaic problem for K = 2 (Wu et al. gave a similar result (Wu & Gusfield 2007)). Also, Ukkonen designed an O(nK^O(m)) time dynamic programming algorithm for the general (i.e., K ≥ 2) Minimum Mosaic problem (the time complexity was guessed from the description given in the paper and will be justified in Section 5). According to the authors, this latter algorithm does not perform well for a moderate number of founders and/or recombinations.

A different (no restriction on the number of founders) but related problem was introduced by Gusfield (Gusfield 2002). 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 et al. (Rastas & Ukkonen 2007) observed that this latter problem is equivalent for K ≤ √2m to the Minimum Mosaic for genotype recombinants without missing values (i.e., D ⊂ (0, 1, 2)^n), that was proved to be NP-complete in (Lancia et al. 2004).

In (Rastas & Ukkonen 2007), Rastas et al. considered the Minimum Mosaic problem with missing values and mutations (i.e., mismatches between founders and recombinants) leading to a different parsimony criterion. More precisely, for each recombinant r ∈ D, a score κ + κc is computed where (1) k is the number of breakpoints of a recombinant r such that the Hamming distance between r and 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 et al. 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 ⊂ (0, 1, 2)^n) is NP-complete. Finally, they proved that even when missing values are forbidden, the Minimum Mosaic problem for haplotype recombinants (i.e., D ⊂ (0, 1)^n) is NP-complete even for K = 2 since it becomes equivalent to the NP-complete Hypercube Segmentation problem (Kleinberg et al. 1998). One has to notice that this latter result is a bit “artificial”, in the sense that the extra condition that the mutation cost c = 1 (necessary in the proof) roughly corresponds to forbid (and thus ignore) recombinations. Indeed, mutation costs becomes less expensive than a single recombinant event. This is precisely the reason why Ukkonen (Ukkonen 2002) and Wu et al. (Wu & Gusfield 2007) have been able to find a polynomial-time algorithm for the problem without mutation. One of the purpose of our contribution is to provide a better complexity insight of the problem by proving its complexity without relying on a somewhat unrealistic prohibitive mutation cost function.

Several heuristics and lower bounds have been proposed (Roli & Blum 2009, Wu 2010). Wu et al. (Wu & Gusfield 2007) designed an O(nmK) time algorithm for the Minimum Mosaic problem for genotype recombinants without missing values (i.e., D ⊂ (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.

Finally, in (Zhang et al. 2008), 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 table.

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_1s_2...s_n, and two integers i and j, 1 ≤ i ≤ j ≤ n, we denote by s[i, j] the substring s_i...s_j of s. Given a set of haplotype founders f_1, f_2, ..., f_K, each in (0, 1)^n, and an haplotype recombinant r of size n, a segmentation of r out from f_1f_2...f_K is a partition of the interval [1 ... n] = {1, 2, ..., n} into consecutive intervals I_1, I_2, ..., I_k such that, for each 1 ≤ i ≤ k, we have r[I_i] = f_j[I_i] for some j ∈ {1, 2, ..., K}. The cost of the segmentation is k – 1. For example, for r = 001011, f_1 = 000000 and f_2 = 111111, f_1 = f_1[1 ... 2], f_2[2 ... 3], I_3 = f_1[4 ... 4], I_4 = f_2[5 ... 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, ..., f_K is denoted cost(r; f_1, f_2, ..., f_K). (the cost of a best segmentation can be found in O(nK) time (Rastas & Ukkonen 2007)).

The Minimum Mosaic problem can be defined as follows: Given a set of m recombinants D = {r_1, r_2, ..., r_m}, r_i ∈ (0, 1)^n for 1 ≤ i ≤ m, and an integer K, find a set of K founders F = {f_1, f_2, ..., f_K}; f_i ∈ (0, 1)^n for 1 ≤ i ≤ K, such that ∑_{i=1}^m cost(r_i; f_1, f_2, ..., f_K) is minimized.

4 Hardness result for the Minimum Mosaic problem

In this section, we prove the Minimum Mosaic problem for an unbounded K to be NP-complete without
relying on unrealistic prohibitive mutation costs (as done in (Rastas & Ukkonen 2007)). 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 $D = \{r_1, r_2, \ldots, r_m\} \subset \Sigma^n$ where $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ is any alphabet on $k$ symbols. Then, take any encoding of the symbols in $\Sigma$ by binary strings of length $\lceil \log_2 k \rceil$. In other words, let $\delta : \Sigma \mapsto \{0, 1\}^k$ be any injection from $\Sigma$ to $\{0, 1\}^k$ with $k' = \lceil \log_2 k \rceil$. Once such an encoding has been fixed, we can extend $\delta$ to get an injective function which maps every string $r_i$ over $\Sigma$ into a binary string $\delta(r_i)$ of length $\lceil \log_2 |r_i| \rceil$. Notice that any feasible solution $(f_1, f_2, \ldots, f_K)$ for the instance $(K, r_1, r_2, \ldots, r_m)$ naturally maps into the feasible solution $(\delta(f_1), \delta(f_2), \ldots, \delta(f_K))$ for the instance $(K, \delta(r_1), \delta(r_2), \ldots, \delta(r_m))$, and the cost remains unaffected.

Let us show that the converse is also true. To do so, we claim that, given any feasible solution $(f'_1, f'_2, \ldots, f'_K)$ for the instance $(K, \delta(r_1), \delta(r_2), \ldots, \delta(r_m))$, 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 $\Sigma$. In other words, we can assume that $f'_i = \delta(f_i)$ for $i = 1, 2, \ldots, K$. 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 $f'_1$ is not cleaned, that is, $f'_1$ does not belong to the image of map $\delta$. Then there exists some $1 \leq j \leq n$ such that $f'_1[k'(j - 1) + 1, k'] \not\in \delta(\sigma) : \sigma \in \Sigma$]. In this case, where $\Sigma$ is any symbol in $\Sigma$ such that $\delta(\sigma)$ has a longest suffix in common with $f'_1[k'(j - 1) + 1, k']$, we modify $f'_1$ precisely on the interval $[k'(j - 1) + 1, k']$, and, more precisely, by replacing it by the string $\delta(\sigma)$. Considering the way the procedure for finding a minimum cost segmentation operates (e.g. Algorithm 1), it is easy to see that this modification does not increase the cost. \[\Box\]

**Algorithm 1** Algorithm for finding a minimum cost segmentation: $F_S(\sigma; f_1, f_2, \ldots, f_K)$

1. Let $i = |\sigma|$
2. If $i = 0$ then return
3. end
4. if $\sigma$ is the smallest positive natural such that $r_1[j, i] = f_k[j, i]$ for some $k$
5. let $j$ be the smallest positive natural such that $r_1[j, i] = f_k[j, i]$ for some $k$
6. if $j > i$ then
7. return "No production exists"\[\Box\]
8. end
9. return $F_S(\sigma; f_1[1, j - 1], f_2[1, j - 1], \ldots, f_K[1, j - 1]) + [j, i]$

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 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.

Let us first remark that it is easy to provide a polynomial-time reduction from any instance containing forced founders to an instance without ones. Indeed, for each forced founder $f^j$, add $n m$ copies of $f^j$ in the recombinants set $D$. It is clear that in a solution, one has to include $f^j$ 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 $f^j$ to get this property.

Thus, for the purpose of the reduction, we can without loss of generality assume to have instances of the form $(K, D, F^j)$ such that $D = \{r_1, r_2, \ldots, r_m\} \subset \Sigma^m$, $F^j = \{f_1, f_2, \ldots, f_m\} \subset \Sigma^m$ and $m = m' + m''$. Let us now provide a reduction from the NP-complete VERTEX-COVER problem: Given a graph $G = (V, E)$ such that $|V| = n_G$, $|E| = m_G$ and an integer $k_G$, decide whether there exists 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. We define $m = C n_G + 3$ recombinants, each of length $n = 6 m_G$ built as follows ($C$ is a constant defined later):

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

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

- $F^j_1 = (ZZZZZZ)$
- $F^j_{t_i} = Z^3 X_{t_j} X_{t_i} X_{h_i} \times Z^3(2 m_G - t - 1)$, 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.

**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 $C(n_G - k_G)(2m_G - 1) + 6 m_G$.

*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 $r_{c_j} \in V' \cap e_j$ (such an index exists since $V'$ is a vertex cover). By definition, $c_j \in \{t_j, h_j\}$. Let $\tau = \{(t_j, h_j)\} \cap V'$. We construct our solution with the following set $F^i$ of $K'' = k_G + 1$ founders:

- $F_0^i = \prod_{j=1}^{n_G} (WWX_{t_j} X_{h_j} WW)$
- $F_{t_i}^i = (X_{t_j} X_{t_i} X_{h_i} X_{h_j})^{m_G}$, for each $v_i \in V'$.

Let us now compute the corresponding cost:

- $\text{cost}(r_1; F^i \cup F^j) = 2 m_G$ since we have to switch from $F_0^i$ to $F_1^j$ and from $F_1^j$ to $F_0^i$ for each $m_G$ blocks of length 6,
cost(r; F’ ∪ F) = 2mc, and cost(r; F’ ∪ F) = 2mc. For each 1 ≤ j ≤ mc, since V’ is a vertex cover, F’ and F will prevent a breakpoint between the X_{ij}’s (and between the X_{ij}’s) since c_j ∈ {t_1, t_2, ..., t_{mc}} \{c_j\}. Therefore, by switching from F’ to F_{c_j} or F_{c_j} and back to F’ for each mc blocks of length 6, it only induces 2 breakpoints for each block.

• for 1 ≤ j ≤ C, cost(r; F’ ∪ F) = 0 for each v_i ∈ V’ whereas cost(r_{ij}; F’ ∪ F) = C(2mc - 1) for each v_i ∉ V’. Considering each recombinants r_{ij}, such that v_i ∉ V’, by switching from F_{r; ij} to 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 (2mc - 1) each.

On the whole, Σ_{i=1}^{n_{G}} Σ_{j=1}^{C} cost(r_{ij}; F’ ∪ F) = 2mc + (2 × 2mc) + C(n_{G} - k_{G})(2mc - 1) = C(n_{G} - k_{G})(2mc - 1) + 6mc.

□

We now turn to considering the reverse direction.

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})(2mc - 1) + 6mc, for any C > 6m, then G admits a vertex cover of size at most k_{G}.

Proof: Let us prove that the cost of recombinants r_{ij}, 1 ≤ i ≤ n_{G}, 1 ≤ j ≤ C, for any solution is greater than C(n_{G} - k_{G})(2mc - 1) - 1. Note first that, only considering the set of forced founders F’, each r_{ij}, 1 ≤ i ≤ n_{G}, 1 ≤ j ≤ C, has a cost(r_{ij}; F’) = C(n_{G} - k_{G}). Indeed, given a recombinant r_{ij}, one has to switch from F_{ij} to F’ for 0 ≤ t ≤ 2mc - 2.

Considering now both F’ and F, if the set of K’ = k_{G} + 1 founders is composed of exact copies of K’ different recombinants r_{ij}, i_1, i_2, ..., i_{k_{G} + 1} (for some 1 ≤ i_1, i_2, ..., i_{k_{G} + 1} ≤ n_{G}, and 1 ≤ j_1, j_2, ..., j_{k_{G} + 1} ≤ C), then Σ_{i=1}^{n_{G}} Σ_{j=1}^{C} cost(r_{ij}; F’ ∪ F) = C(n_{G} - k_{G} + 1)(2mc - 1). Hence, r_{ij} 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 ≤ t ≤ mc - 1. Then, for any 0 ≤ t ≤ mc - 1, there will exist among the recombinants {r_{ij}, r_{i_1}, r_{i_2}, ..., r_{i_{k_{G} + 1}}} one recombinant – say r_{ij} – that will induce a switch (for each of its C copies) from a founder of F to F’ for 0 ≤ t ≤ 2mc - 2 (resp. F_{ij}, i_{1}, i_{2}, ..., i_{k_{G} + 1} + 2) 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 2Cmc. In the best case, one should only “sacrifice” one of the K’ founders (which was only allowing a gain of 2Cmc - 1 breakpoints). On the whole, Σ_{i=1}^{n_{G}} Σ_{j=1}^{C} cost(r_{ij}; F’ ∪ F) ≥ C(n_{G} - k_{G})(2mc - 1). Moreover, considering the objective cost (i.e. C(n_{G} - k_{G})(2mc - 1) + 6mc) and that C > 6m, the following is enforced:

1. Considering any position, each of the K’ founders has a different letter.
2. A founder with a letter X_{a} in a given position contains only letters X_{b} in the next positions until a letter W or the end of the founder is encountered.

3. All the W letters occur in the same founder.

Let us now prove each of these points.

Point 1. 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}, r_{i_2}, ..., r_{i_{k_{G} + 1}}} which leads to a cost above the objective.

Point 2. By contradiction, in an optimal solution, suppose there is a founder with a letter X_{a} at position k followed by a letter X_{b}. Then, due to Point 1, there should not exist another founder such that X_{c} is at position k nor X_{d} at position k + 1. Therefore, there is a breakpoint between positions k and k + 1 in all r_{ij} (resp. r_{ij}); that is 2C breakpoints. Consider now changing X_{c} into X_{d} in the corresponding founder. Then it will induce at least C less breakpoints, leading to a better solution; a contradiction.

Point 3. 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_{i}, i ≠ j. By Point 1, there is no W letter at position 6t + 2 (resp. 6t + 1) in founder F_{j} (resp. F_{i}). Thus, there is a breakpoint in the recombinant r_{ij} between positions 6t + 1 and 6t + 2. Consider now swapping F_{i}.(6t + 1) and F_{j}.(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 3 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_{a} (resp. X_{b}) 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_{ij}, for 1 ≤ x ≤ C, and one breakpoint between positions 6t + 6 and 6t + 7 in all r_{ij}, for 1 ≤ x ≤ C. Consider now swapping F_{i}.(6t + 4) and F_{j}.(6t + 4) and replacing that X_{a}’s of F_{i} by X_{b}. It will induce at least C less breakpoints, leading to a better solution; a contradiction.

In the following, let us assume that F_{0} denote the founder containing the W letters; the other founders are referred 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}, ..., F_{K’}} corresponds to a vertex cover in G. Let V’ \subseteq V be defined as follows. For each founder F_{i}, 1 ≤ i ≤ K’, if F_{i} is the exact copy of the recombinant r_{ij}, then add r_{ij} to 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}, v_{x}) \in E, and both (v_{h}, v_{x}) ∩ V’ = \emptyset. Therefore, there is at least a breakpoint between X_{h} and X_{x} in r_{3} or between X_{h} and X_{x} in r_{3} since neither h nor x appears in {F_{1}, F_{2}, ..., F_{K’}}, and F_{0} cannot contain at most one of the two in the corresponding positions. It is worth noticing that none of the forced founders cannot prevent thses breakpoints since there never have the same letters at position 6t + 3 and 6t + 4.

□

Combining Lemma 2 and Lemma 3 we obtain the following result.

Proposition 4 Minimum Mosaic given haplotype recombinants with no missing values is NP-Complete.
W not on the same founder

a) ...(XuXuX#, u XuXuXu)(Xu... 
...(XvXvX#, v XvXvX#)

b) ... when con-

... being considered then all the segments starting at a given position,
one should be able to find a distribution of the seg-

\[ \text{hal-00512458, version 2 - 14 Oct 2010} \]

... the minimum number of breakpoints considering all partitions of any column

... the minimum number of breakpoints inducing. One has to store in a
dynamic table, the minimal cost of any

... partitions for the column. On the whole, since (1) there are

... and (2) one has to compute

... minimal mosaic is bounded, then the

... of the column. The central recurrence is based on the fact that

... of size \( i \), of the

... by b) and c) are denoted by \( \# \).

\[ \text{5 Exact algorithms for Minimum Mosaic} \]

In this section, we will provide exact algo-
rithms considering a variant of the Minimum Mosaic/Segmentation problems where extra informa-
tions (e.g. position, number, \ldots) on the breakpoints are given.

Let us first give a complexity study of the gen-
eral solution (Section 4) proposed by Ukkonen in (Ukkonen 2002). Indeed, the corresponding solution shows that, once the number of founders (i.e. \( K \)) is bounded, then the Minimum Mosaic problem is

... and a set of \( \{ I_1, I_2, \ldots, I_K \} \) such that \( I_1 = [n, \ldots, n] \), \( 1 \leq i \leq K \)

... LCP = 0, \( F' = \text{null} \)

... the least empty position in \( F \);

greater than \( L_y^x \)

... \( I_y \) and \( F' \) share a common prefix then
then

... for each \( F_i \) in \( F \) do

... Z ← the least empty position in \( F_i \);

greater than \( L_y^x \)

... if \( F_i[L_y^x, Z] \) and \( I_y \) share a common prefix then

... end if

... end for

... if \( F' = \text{null} \) then

... F'[L_y^x, R_y^x] ← I_y

... end if

... end for

... Exit with error

... \( I_y \) and \( L_y^x \) are empty then

... F'[L_y^x, R_y^x] ← I_y

... end if

... end for

... exit

... return F

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

... founders to \( K \) founders. To do so, the algorithm computes the

... founders from left to right using the available seg-

... segments in \( L \). For each available segment \( I_y^x \), the algo-

... the algorithm tries to find a founder such that the position \( L_y^x \)

... 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 ex-

... such a solution then any segments can be placed

... one should be able to find a distribution of the seg-
ments among the $K$ founders. This ensures that when considering $I^F_{y}$, the choice of the founder for $I^F_{x}$ will not interfere with placement of $I^y_{x'}$. Indeed, either in the solution (1) they were on the same founder and thus will have a common prefix or (2) were not in the same founder. This leads to an overall $O(BKL)$ time-complexity algorithm where $L$ is the length of the longest segment.

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

Finally, we give an algorithm with a lower complexity 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 $\binom{m_d}{B} = \binom{K + B}{B} = (K + B)^B \binom{B}{B}$ different configurations. For each one, run the exact $O(nK^{2m_B})$ 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 bounded 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 for an unbounded number $K$ of founders. When $K$ is bounded 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 bounded, the polynomial-time complexity when $K = 2$, and the NP-Completeness when $K$ is unbounded. It is also widely open whether the problem admits some PTAS.

References


Lancia, G., Pinotti, M. C. & Rizzi, R. (2004). ‘Haplo-