Comparison of Minisatellites

SŁverine Bérard
LIRMM, UMR CNRS 5506
161, rue Ada, F34392 Montpellier Cedex 5
France
berard@lirmm.fr

Éric Rivals
LIRMM, UMR CNRS 5506
161, rue Ada, F34392 Montpellier Cedex 5
France
rivals@lirmm.fr

ABSTRACT

In the class of repeated sequences that occur in DNA, minisatellites have been found polymorphic and became useful tools in genetic mapping and forensic studies. They consist of a heterogeneous tandem array of a short repeat unit. The slightly different units along the array are called variants. Minisatellites evolve mainly through tandem duplications and tandem deletions of variants. Jeffreys et al. devised a method to obtain the sequence of variants along the array in a digital code, and called such sequences maps. Minisatellite maps give access to the detail of mutation processes at work on such loci. In this paper, we design an algorithm to compare two maps under an evolutionary model that includes deletion, insertion, mutation, tandem duplication and tandem deletion of a variant. Our method computes an optimal alignment in reasonable time; and the alignment score, i.e., the weighted sum of its elementary operations, is a distance metric between maps. The main difficulty is that the optimal sequence of operations depends on the order in which they are applied to the map. Taking the maps of the minisatellite MSY1 of 609 men, we computed all pairwise distances and reconstruct an evolutionary tree of these individuals. MSY1 (DYF155S1) is a hypervariable locus on the Y chromosome. In our tree, the populations of some haplogroups are monophyletic, showing that one can decipher a micro-evolutionary signal using minisatellite maps comparison.

Categories and Subject Descriptors

G.4 [**Mathematical software**]: [algorithm design and analysis, efficiency, certification and testing]; I.5.3 [**Pattern recognition**]: Clustering—*algorithm, similarity measures*; I.5.4 [**Pattern recognition**]: Applications—*text processing*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*combinatorial algorithms*; J.3 [**Life and Medical Science**]: [biology and genetics]

General Terms

Algorithms, measurement

Keywords

Alignment, bioinformatics, dynamic programming, evolution, minisatellite, overlap graphs, sequence comparison, tandem repeats

1. INTRODUCTION

Repeated sequences represent a large part of eukaryotic genomes. Among repeats, the class of tandem repeats, whose duplicated units are adjacent along the chromosome, have also been found in other species including bacteria. This class is subdivided further by decreasing order of repeat unit size into satellites, mini-satellites and micro-satellites.

Minisatellites (ms) consist of tandem arrays of short repeat units. Tandem repeats classifications vary and the repeat unit length of ms is considered to be between 7 and 100 bp [15] or 10 and 50 bp [11]. Due to a lack of a precise definition of ms, these structures are usually not annotated in sequence data. Nevertheless, amongst the community of biologists there has grown a large interest in ms because of their polymorphism; Variable Number of Tandem Repeats (VTNR) is a synonym for polymorphic ms. This variability is due to tandem duplication, an event that adds a duplicated copy next to the original one, and to the reverse event, tandem deletion. Slippage of DNA polymerase during replication and unequal crossing-over have been hypothesized as mechanisms for these events. As for microsatellites, length variations of ms have been found to be involved in several diseases like diabetes, epilepsy and cancer and there is evidence for other contributions to genome function (see [6, 15]).

Specific studies reveal that most human ms contain subtly varying repeat units [9]. In 1991, Jeffreys and colleagues designed a PCR reaction to type Minisatellites Variant Repeat, the MVR-PCR [10]. Each different unit is a variant. This method provides the sequence of variants along the array in a digital code, where each variant is encoded by a symbol. Such sequences are called **internal** or **minisatellite maps**. This technology has led to major improvements in the knowledge of ms instability processes, among which are: the model of mutation initiation by double strand breaks, the role of flanking sequences, the differences between meiotic and somatic mutations and the phenomenon of polarized variability (see [9, 15] for reviews).

Because of their length variability, minisatellites have proven useful in genetic mapping, forensic studies, and the exploration of genetic diversity and population structure. But ms maps give access to the mutation processes of ms at a more detailed level. For instance, they allowed Armour et al. to investigate the Out-of-Africa hypothesis of human origin [2].

The human minisatellite MSY1, the most variable locus on the Y chromosome, is an ideal tool to investigate the evolution of this haploid chromosome and decipher paternal lineages in human [11]. In [11], the authors typed by MVR-PCR the MSY1 maps of 690 men sampled in various human populations. Their aim was to survey MSY1 diversity in haplogroups and populations. They studied the average maps diversity in some populations, but the lack of a computer-based comparison method impeded a more in-depth study (like reconstructing an evolutionary tree of these individuals.)

To fully exploit the evolutionary information contained in these sequences of variants, it is critical to be able to compare automatically ms maps. In this paper, we provide an alignment algorithm which considers the events of tandem duplication and deletion of a variant and is specific for ms maps. We hypothesize a symmetrical single-step evolutionary model for ms, which we present in Section 2. In Section 3, we detail the algorithm. In Section 4, we apply our method to compare the MSY1 maps from [11] and reconstruct an evolutionary tree from the resulting distance matrix.

Notation. As minisatellite maps can be modeled by sequences of symbols, also called strings, we introduce a notation for strings. Let Σ be a finite alphabet of variants and σ its size. A map s of length n is a string of n symbols of Σ indexed from 1 to n. We denote the *length* of s by $|s|$ and its i-th *symbol* by $s[i]$ for all i with $1 \leq i \leq n$. For any integers i, j, $1 \leq i \leq j < n$, $s_j^i := s[i] \dots s[j]$ is called a *substring* of s. We denote the concatenation of two strings r and s by $r.s$. Throughout the article, let r, s be two maps over Σ of length m and n, respectively.

2. SINGLE-STEP EVOLUTIONARY MODEL FOR MINISATELLITES

In this section we describe our evolutionary model for minisatellites and introduce the notion of arches.

Our model considers five types of evolutionary events on variants: mutation, insertion, deletion, amplification and contraction. Mutation, insertion and deletion are the same events as those considered traditionally in sequence alignments except that they apply to a variant instead of a single residue. Here is an illustration on the sequence *abc*:

delete b:	$abc \rightarrow ac$
insert d at pos. 3:	$abc \rightarrow abdc$
mutate b in d:	$abc \rightarrow adc$

The two specific events are amplification and contraction (These are shorter names for tandem duplication and tandem deletion.) For example, the sequence *abc* undergoes an amplification of variant b and then its contraction:

amplification:	$abc \rightarrow abbc$
contraction:	$abbc \rightarrow abc.$

In the map s, the amplification of the variant $s[i]$ results in the insertion of a new copy of $s[i]$ at position $i+1$. The contraction is the dual operation: if $s[i]$ occurs at two successive positions, i.e., $s[i] = s[i+1]$, the contraction at position $i+1$ removes $s[i+1]$. Our model is termed single-step because amplification, resp. contraction, adds, resp. removes, a single variant at a time. A future direction of research is to consider a multiple-step model where, e.g., a triplication or a quadruplication of a variant may happen in a single event.

To complete our model, we need a quantitative criterion to judge map similarity. For this, each operation is associated with a real positive cost. A sequence of events that transforms s into r is called an alignment. The alignment cost is the sum of its operations costs where aligning two identical variants costs zero. We designate each cost by the uppercase initial of the corresponding event: M, I, D, A and C. Here, we consider a symmetrical model where dual events have the same cost $I = D$ and $A = C$. The observed much higher relative frequency of amplifications and contractions compared to other events is translated by the fact that they have lower costs: $A, C < M, D, I$. To unify the notation for a mutation and a match we use $M(a, b)$ that is equal to 0 if $a = b$ and to M otherwise. For the sake of simplicity, we consider that all possible mutations cost the same regardless the nucleotidic sequences of the variants. This is a reasonable assumption for the application to minisatellites since variants are long and differ from each other by a few base pairs (see the case of MSY1 in Section 4.) As in general, a deletion can also be obtained by a mutation plus a contraction and an insertion by an amplification plus a mutation, we have either ($D > M + C$ and $I > A + M$) or ($D \leq M + C$ and $I \leq A + M$). Without loss of generality we assume the first hypothesis (denoted H1). This influences the cost of arch generations/compressions (see Lemma 1) and modifies slightly the algorithm. With H1, at any position except the first one, a variant can be "inserted" by an amplification+mutation, or "deleted" by a mutation+contraction. We denote these operations by A_M and M_C, resp. and they cost $A+M$ and $M+C$, resp. Under these conditions, the alignment cost is a metric; this is important when reconstructing an evolutionary tree from maps data. The proof that the alignment distance is a metric is in the appendix.

2.1 The Concept of Arch

Let us look step-wise at an artificial example of evolution of the same minisatellite in two individuals (Figure 1). The alphabet of variants is $\{a, b, c, d, e\}$, the ancestor state has map *aaaaa*, the resulting maps in Individuals 1 and 2 are $r :=$ *aeaaa* and $s :=$ *aaabbcbddba*, respectively. An alignment is given Figure 2. In Individual 2, the variant b is amplified five times. These amplified variants further mutate and then undergo additional amplifications. This results in a substring whose variants all come from the same ancestor variant, which we call the **seed** of the substring. We can delimit such a substring by the fact that its extremal variants are identical. In our example, the substring of s is *bbcbddb* from position 4 to 10, its seed b is at position 4. During the evolution of this substring, the final variant at each position

does not appear in the order of the sequence: at some stage, position 8 has still the ancestor state b and not its final state d, while position 10 is already a b. Thus, such substrings can be obtained by many series of operations, but the optimal way may be order-dependent. If we compute incrementally alignment for longer and longer prefixes, i.e., adding one operation at a time, we cannot find the optimal order of events. For our example substring, if the 9th position first becomes a d, it is not possible to obtain a b at the 10th position by an amplification. So, to recover the optimal series of operations that leads to such a substring, we need to consider it as a whole and not each variant one after the other. This leads to the notion of arches.

Definition 1. *Let s be a map of length n and i, j be two integers such that $1 \leq i < j \leq n$. s_j^i is an* ***arch*** *of s if $s[i] = s[j]$.*

In an arch, not all variants need to be the same and an arch may recursively contain other **inner-arches** with different seeds as its, like *dd* in *bbcbddb*. Indeed, once a position with a seed has been mutated into a non-seed variant, it may undergo an amplification which creates an inner-arch.

Individual 1

Event	Sequence
	a a a a a
mut	a **e** a a a

Individual 2

Event	Sequence
	a a a a a
mut	a a a **b** a
5*amp	a a a b **b b b b b** a
mut	a a a b b **c** b b b a
mut	a a a b b c b **d** b a
amp	a a a b b c b d **d** b a

Figure 1: Example of evolution of a minisatellite in 2 individuals.

In the alignment Figure 2, one sees that none of the positions corresponding to *bbcbddb* in s exist in r except the seed position. To mimic evolution, we want to align an arch of s with a single variant of r, the one at the original seed location. In other words, we want to align this seed with a single variant of r and expand additional variants of the arch from its seed. We may see the advent of an arch as a single evolutionary step with a special cost function. Depending on the direction we observe evolution, from s to r or from r to s, we named this an arch **compression** or **generation**. In our dynamic programming (DP) approach, arch generation/compression are viewed as single operations that express dependencies between two non-neighboring entries of the DP matrix (see Figure 4.)

Generation and compression of an arch are symmetrical. Generation corresponds to a series of amplifications and mutations (this depends on hypothesis H1, see above) that gives rise to the associated substring from the seed variant, while compression is a series of contractions and mutations that rewinds the substring into its seed. Suppose an arch in s, as no character from r plays a role in such series of events, its optimal cost can be computed independently of r. We denote by $G(t)$ and $K(t)$ respectively the generation and

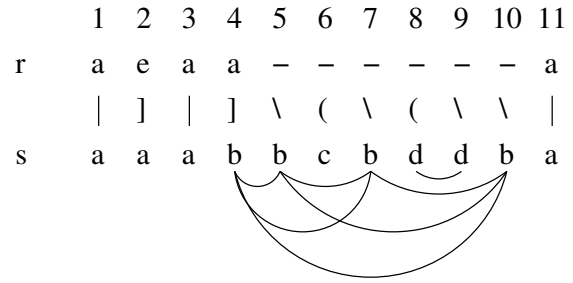

Figure 2: Alignment of the maps of individuals 1 & 2 (see Figure 1). The arch *bbcbddb* and its inner-arches are drawn by curved lines under s. In the middle line, '|', ']', '\', '(' denote resp. a match, a mismatch, an amplification, and an amplification+mutation.

compression costs of an arch t. From now on, we will consider only arch compressions because of the symmetry. In Section 3.2, we explain how to compute the cost of an arch compression and show that it provides an economy of at least M over all other series of operations.

A question arises: as for arches, do we need to consider sequences of operations in which order matters for a submap whose extremal variants are not identical? For we want to recover all possible most parsimonious histories, the answer is yes. It leads to the definition of a **phantom arch**.

Definition 2. *Let s be a map of length n and i, j be two integers such that $1 \leq i < j \leq n$. s_j^i is a* ***phantom arch*** *of s if $s[i] \neq s[j]$.*

To distinguish between the two classes of arches, we qualify a non-phantom arch as **visible**. We show in Lemma 2 that phantom arch compressions/generations are optimal depending with which variant of the other map the seed is aligned. Compared to visible arches, the compression of a phantom arch requires an additional mutation. In a case where considering a phantom arch is optimal, it exists another series of operations that is also optimal.

2.2 Order of Events in an Alignment

The discussion on arches and the example of Figure 1 raise several issues. First, in a single column of an alignment, several events, usually no more than two, may occur. This is the case of column 6 in Figure 2 which is created by an amplification of the neighboring variant 'b', and a mutation is then needed to transform this b into c. The two operations on the same position are illustrated in the detail of the evolution of Individual 1 in Figure 1. Second, it is clear from this example that the order of events matters. Another example is the following: the amplification that creates the b at position 10 has to be performed before the mutation that turns the b at position 8 into a d. In mathematical terms, we say that the operations are not commutative. Third, alignments do not represent well all these informations that are contained in an history. Actually, **listings**, as defined in [13], are better representations for our evolutionary model. Given two maps, our algorithm does recover all possibles optimal listings. Fourth, the non-commutativity forces us to consider

the order of operations to compute the optimal compression or generation of an arch. It explains why we need a specific procedure to compute an arch compression/generation cost, and why this procedure not only consider the arch variant after variant, but uses the submap of the arch as whole. In fact, the non-commutativity increases the computational difficulty.

2.3 Related Works

Traditional global alignment methods do not consider tandem duplication and deletion. The first work that extended the evolutionary model to include such events is the sequence alignment algorithm under the Duplication, Substitution, Indel or DSI-model from Benson [3]. It aims at aligning sequences that may contain tandem repeats. To take into account possible duplication events, the algorithm must identify local repeats in the sequences, which leads to high complexities. Our approach is different since it compares already identified minisatellites. In practice, there exist in-silico and experimental methods for minisatellites detection [4]. Moreover in our model, possible duplicated patterns are the variants, which are known in advance. Another and fundamental difference lies in the treatment of arches. In the DSI model, the cost of a duplication is computed by comparing the new unit to a unit in the other sequence. In a case like individual 2 in our example, the variants b or c, which do not occur in individual 1, would be considered as duplications of the variant a of individual 1. Thus, the DSI model does not reflect the fact that a duplicated unit and its copies are in the same sequence, and cannot cope with the evolution of an arch. On the other hand, Benson provides algorithms for both the single- and multiple-step models ([3]). Our approach provides more details on the mutation process. In some way, one could say that it deciphers partly the history of each map to better compare it with the other map. This is related to the problem of reconstructing the duplication history of a tandem repeat sequence. This problem has been proposed by Benson & Dong [5] and investigated recently for tandemly repeated genes in [7, 14].

3. THE ALIGNMENT ALGORITHM

In this work, we want to compute the optimal global alignment of two minisatellite maps under the single-step model. In Section 3.1, we present our dynamic programming approach assuming that arch generation and compression are single operations. In Section 3.2, we show how to compute the cost of an arch generation or compression for which we need a maximal set of compatible arches. We describe a preprocessing that computes these sets for all arches considered during computation. We finish by establishing the algorithm complexity.

3.1 The Dynamic Programming Computation

We denote by $\mathcal{A}$ the $(n+1)\times(m+1)$ dynamic programming matrix. Its lines and columns are indexed from 0 to n and 0 to m resp. Entry $\mathcal{A}(i,j)$ gives the alignment distance between the prefixes of s and r of length i and j resp., i.e., between s_i^1 and r_j^1, and entry $\mathcal{A}(n,m)$ gives the distance between maps $s := s_n^1$ and $r := r_m^1$. Remember that maps s and r are indexed respectively from 1 to n and from 1 to m. Figure 3 displays the recurrence formula under the hypothesis H1.

First, note that deletion and insertion are used only to compute $\mathcal{A}(1,0)$ and $\mathcal{A}(0,1)$. This is because an amplification is not allowed in an empty string and a contraction is not allowed in a one-variant string. For all other entries, when an insertion or a deletion is needed A_M or M_C are used because of H1. The DP recurrence takes the minimum between terms that express the dependencies shown in Figure 4. The first five terms are due to elementary operations and the others to arch generations and compressions. The latter terms depends on the beginning position l', resp. l, of the arch being generated, resp. compressed. The set of positions for l', resp. l, are given and thus each line represents multiple terms. Compressions, resp. generations, of phantom arches are divided into two subcases depending on which extremal variant match the variant of the other map; this requires that $j > 0$, resp. $i > 0$. Further explanations of the lines concerning phantom arches are given in the proof of Lemma 2. An arch of length 2 can be compressed by a single contraction and a phantom arch by a mutation+contraction. Symmetrically, generations of arches of length 2 are reducible to a single elementary operation. Such arch compressions and generations are therefore not considered in the recurrence. This is why the ending position of an arch is required to be in $[1, i-2]$ with $i > 2$ for compressions and in $[1, j-2]$ with $j > 2$ for generations. A match or a mutation cannot occur in the first row or first column of the matrix; it follows that $i > 0$ and $j > 0$. For any term involving a contraction, resp. an amplification, it is required that there are two adjacent identical variants, which is possible only if $i > 1$, resp. $j > 1$.

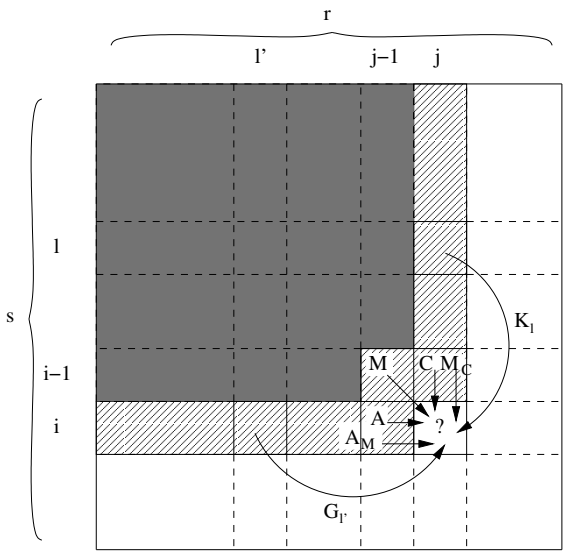

Figure 4: Dependencies in the dynamic programming matrix.

3.2 Costs of Arch Generation and Compression

The arches considered in the main recurrence are sequences of variants whose extremal variants are identical. One extremal variant is assumed to be the ancestor variant of the whole arch. Let us consider first the arch compression of a **simple** arch of length $k > 1$: an arch that contains no inner-arches. There are two ways to rewind this arch. These possibilities are illustrated in Figure 5. Option (i) mutates and

Initialization: $\mathcal{A}(0,0) := 0$, $\mathcal{A}(1,0) := D$ and $\mathcal{A}(0,1) := I$.
For all other entries the recurrence is:

$$\mathcal{A}(i,j) := \min \begin{cases} \mathcal{A}(i-1,j-1) + M(s[i],r[j]) & \textit{Mutate or Match} \text{ if } i>0 \text{ and } j>0 \\ \mathcal{A}(i-1,j) + C & \textit{Contract if } i>1, s[i-1]=s[i] \\ \mathcal{A}(i-1,j) + M + C & \textit{Mutate \& Contract} \text{ if } i>1 \\ \mathcal{A}(i,j-1) + A & \textit{Amplify} \text{ if } j>1, r[j-1]=r[j] \\ \mathcal{A}(i,j-1) + A + M & \textit{Amplify \& Mutate} \text{ if } j>1 \\ \mathcal{A}(i,l') + G(r_j^{l'}) & \textit{Generate arch} \\ & \text{if } j>2, \forall l' \in [1,j-2] : r[l'] = r[j] \\ \mathcal{A}(i,l') + G(r[j].r_j^{l'+1}) & \textit{Generate phantom arch (type 1)} \text{ if } j>2, i>0, \\ & \text{and } s[i]=r[j], \forall l' \in [1,j-2] : r[l'] \neq r[j] \\ \mathcal{A}(i,l') + G(r_{j-1}^{l'}.r[l']) + M & \textit{Generate phantom arch (type 2)} \text{ if } j>2, i>0, \\ & \text{and } s[i]=r[l'], \forall l' \in [1,j-2] : r[l'] \neq r[j] \\ \mathcal{A}(l,j) + K(s_i^l) & \textit{Compress arch} \\ & \text{if } i>2, \forall l \in [1,i-2] : s[l]=s[i] \\ \mathcal{A}(l,j) + K(s[i].s_i^{l+1}) & \textit{Compress phantom arch (type 1)} \text{ if } i>2, j>0, \\ & \text{and } s[i]=r[j], \forall l \in [1,i-2] : s[l] \neq s[i] \\ \mathcal{A}(l,j) + K(s_{i-1}^l.s[i]) + M & \textit{Compress phantom arch (type 2)} \text{ if } i>2, j>0, \\ & \text{and } s[l]=r[j], \forall l \in [1,i-2] : s[l] \neq s[i] \end{cases}$$

Figure 3: Recurrence.

contracts the end variant, then removes the $k-2$ internal variants at optimal cost R, while option (ii) first removes the $k-2$ internal variants at optimal cost R and contracts the now adjacent extremal variants in one. In both cases, we are left with a single seed variant. Option (i) costs $R+M+C$ and option (ii) $R+C$; thus, option (ii) is optimal. The economy obtained by choosing the arch compression, i.e., option (ii), is M.

(i)

Event	Sequence				
	$s[i-k+1]$	$s[i-k+2]$	$\ldots$	$s[i-1]$	$s[i]$
mut	$s[i-k+1]$	$s[i-k+2]$	$\ldots$	$s[i-1]$	$s[i-1]$
con	$s[i-k+1]$	$\boxed{s[i-k+2] \quad \ldots \quad s[i-1]}$			
remA	$s[i-k+1]$				

(ii)

Event	Sequence				
	$s[i-k+1]$	$\boxed{s[i-k+2] \quad \ldots \quad s[i-1]}$			$s[i]$
remA	$s[i-k+1]$	$s[i]$			
con	$s[i-k+1]$				

Figure 5: Two ways to rewind the arch s_i^{i-k+1}, with $i>k$. Note that $s[i-k+1]=s[i]$ and remA is an abbreviation for "remove all the internal variants (those in the rectangle) at optimal cost R".

Now, consider a complex arch of length k; it encloses inner-arches. The optimal compression would be to recursively use option (ii) to compress all arches, i.e., the outer-arch and the inner-arches. But not all arches can be compressed. Indeed, when two arches, say u, v, overlap, the seed of u is an internal variant of v and it is not possible to compress v and u, since it would require deleting the seed of u which cannot then be contracted when compressing u. So the optimal compression of a complex arch should maximize the number of arches that are compressed. For this we need to compute the maximal set of arches that can be compressed together; we term them **compatible**. Note that there is no phantom inner-arch.

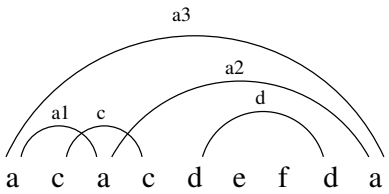

Figure 6: Arches a_1, c and d are simple, while a_2 and a_3 are complex. The following pairs of arches are incompatible: $(a_1,c),(a_1,a_3),(c,a_2),(a_2,a_3)$, while $(a_3,c),(a_3,d),(a_1,a_2),(a_1,d),(a_2,d),(c,d)$ are compatible.

Definition 3. Arch incompatibility *Two arches are incompatible if they overlap by strictly more than one variant and are not contained in each other, or if they share the same first or the same last position. Compatible is the opposite of incompatible.*

Simple, complex, compatible and incompatible arches are illustrated by Figure 6.

Lemma 1. Arch Compression Cost *Let u be an arch of k variants and p the maximum number of compatible arches among all arches including u. Then the optimal compression cost of u is $(k-1) \times C + (k-1-p) \times M$.*

Proof. In the arch, all variants are contracted except the seed; thus, the costs for contractions is $(k-1) \times C$. Of the two extremal variants of a compressed arch, one is contracted in the seed and the other is the remaining seed. Thus, the number of mutations is at most $k-1-p$. Clearly, each mutated variant is either internal to a compressed inner-arch or a remaining seed after compression of all compatible inner-arches or an unpaired variant (i.e., one that is not the extremal variant of any compatible arch). To see that remember that after compression of all compatible inner-arches a complex arch is a simple arch. Therefore, we need at least $k-1-p$ mutations and their total cost is $(k-1-p) \times M$. This gives a total cost of $(k-1) \times C + (k-1-p) \times M$ as claimed. □

Here we exhibit the conditions under which phantom arch compressions/generations lead to an optimal alignment. We demonstrate the lemma only for phantom arch compression, the case of generation is symmetrical.

Lemma 2. *Let i, j be two integers such that $i > 2$ and $j > 0$. To compute $\mathcal{A}(i,j)$ under hypothesis H1, it is necessary to consider the compression of a phantom arch s_i^l, with $l \in [1, i-2]$, iff $s[i] = r[j]$ xor $s[l] = r[j]$.*

Proof. We examine the several ways to compress a phantom arch s_i^l in a single variant either $s[l]$ or $s[i]$, which is aligned with $r[j]$. There are 3 alternatives:

1. $s[i] = r[j]$ which corresponds to a compression of type 1.
2. $s[l] = r[j]$ which corresponds to a compression of type 2.
3. $s[i] \neq r[j]$ and $s[l] \neq r[j]$ which leads to a non-optimal compression and is not considered at all in the recurrence.

1. $s[i] = r[j]$: We align $s[i]$ with $r[j]$, and there are 2 options
   (a) mutate $s[l]$ in $s[i]$, optimally remove the $i-l-1$ internal variants at cost R, and contract $s[l]$ in $s[i]$. The whole cost is $R + M + C$.
   (b) or mutate and contract $s[l]$ in $s[l+1]$, and optimally remove the $i-l-1$ internal variants at cost R. The cost is $R + M + C$.

   Options (a) and (b) cost the same. However in option (a), the order in which operations are applied matters, like for a visible arch. So, option (a) is what we call phantom arch compression of type 1. In the main recurrence, the mutation is already accounted for in $\mathcal{A}(l,j)$ and the term $K(s[i].s_i^{l+1})$ computes the cost of the arch compression after the mutation; this is why the sequence inside the parentheses begins with $s[i]$ and not with $s[l]$.

2. $s[l] = r[j]$: This case is symmetrical to case 1. where l plays the role of i and vice versa. The cost is $R + M + C$. As for the previous case, the remark on the recurrence is also valid for compressions of type 2, except that the mutation is not accounted for in $\mathcal{A}(l,j)$ since it applies to $s[i]$. This is why a term $+M$ appears on that line in the recurrence.

3. $s[i] \neq r[j]$ and $s[l] \neq r[j]$: We can optimally remove the $i-l-1$ internal variants at cost R, then we have two symmetrical possibilities as we choose $s[l]$ or $s[i]$ to be mutated in $r[j]$ and the other to be mutated and contracted. In both case, the cost is $R + 2M + C$.

As stated in the lemma, compression of phantom arches are optimal iff $s[i] = r[j]$ xor $s[l] = r[j]$. □

Computing the maximal set of compatible arches. As a substring, an arch is associated to an interval of indices. The incompatibility relationship defines an overlap relationship between these intervals (not an overlap+containment relationship). Let $G := (V, E)$ be the graph such that V is the set of intervals associated to arches, and an edge links two intervals if the arches are incompatible. G is an overlap graph. The problem of finding the maximal number of compatible arches is equivalent to finding the max stable set, also known as maximum independent set, of G. By slightly modifying the intervals associated with arches we can enforce that no two intervals have the same endpoints; the number of intervals is then in the order of $\Theta(k^2)$ for an arch of length k. Under this condition, Apostolico et al. [1] reports a $O(|V| \times (d + \log(|V|))$ algorithm for the max stable set of an overlap graph, where $|V|$ denotes the cardinal of V and d the density of the interval model of G. This algorithm also computes for any interval l, the max stable set for the graph G restricted to all intervals included in l. We could therefore get the max number of compatible arches for all visible arches in one run, but not for the phantom arches.

Set of useful arches. Let $1 < i < n$ and $1 < j < m$. We compute the entry $\mathcal{A}(i,j)$ of the matrix with the recurrence. The set of starting positions of visible arches of s ending at position i is $\mathcal{C}_i := \{0 < l < i \ : \ s[l] = s[i]\}$. This set is independent of j and thus, visible arches that need to be evaluated are the same for all entries on line i of the matrix. The set of starting positions of phantom arches is $\mathcal{P}_{i,j} := \{0 < l < i \ : \ s[l] \neq s[i] \text{ and } ((s[i] = r[j]) \text{ or } (s[l] = r[j]))\}$. For the $\mathcal{P}_{i,j}$, the condition $((s[i] = r[j]) \text{ or } (s[l] = r[j]))$ ensures that different positions are considered for different $r[j]$. It means that for all pairs j, j' with $j \neq j'$: $\mathcal{P}_{i,j} \cap \mathcal{P}_{i,j'} = \emptyset$ if $r[j] \neq r[j']$. Moreover, $\mathcal{C}_i$ and $\mathcal{P}_{i,j}$ are disjoint for all $j \ : \ 0 < j \leq m$, and their union is a subset of $[1, i-1]$. This means that there are at most $i-1$ possible arches to consider to compute entries of line i of $\mathcal{A}$, which implies a maximum n^2 arch compressions and symmetrically m^2 arch generations for the whole matrix $\mathcal{A}$.

Computation of the compression cost of a phantom arch going from position l to i, with $0 < l < i \leq n$ in s. The two types of phantom arches are symmetrical; so here and in Lemma 3, we consider only phantom arches type 1. For type 1, the arch sequence considered for compression is not s_i^l, but $s[i].s_i^{l+1}$. I.e., the first variant is changed to $s[i]$, which is also the end variant. This change creates exactly as many arches as there are other occurrences of $s[i]$ in s_i^{l+1}. We show that knowing the maximal set of compatible arches corresponding to s_i^{l+1} we can deduce the set corresponding to $s[i].s_i^{l+1}$.

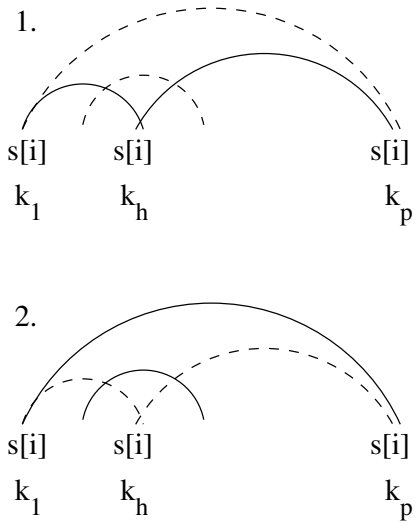

Figure 7: Illustration of the two cases of Lemma 3: plain-line arches belong to the maximum sets while dashed-line arches do not.

Lemma 3. Phantom arch maximal set computation (for type 1). *Let $\mathcal{Y}, \mathcal{Z}$ be the maximal sets of compatible arches of s_i^{l+1}, $s[i].s_i^{l+1}$ respectively. Let $p > 1$ be an integer and $k_1 := l < \ldots < k_p := i$ be the ordered positions of variant $s[i]$ in $s[i].s_i^{l+1}$. For any h, h' in $[1, p]$ such that $h < h'$, let us denote the arch going from k_h to $k_{h'}$ by $H_{k_h,k_{h'}}$. We have two possible cases illustrated in Figure 7:*

1. *Either it exists $h : 1 < h < p$ such that $H_{k_h,k_p} \in \mathcal{Y}$ and then $\mathcal{Z} := \mathcal{Y} \cup \{H_{k_1,k_h}\}$ and $H_{k_1,k_p} \notin \mathcal{Z}$,*
2. *or for all $h : 1 < h < p$ $H_{k_h,k_p} \notin \mathcal{Y}$ and thus $\mathcal{Z} := \mathcal{Y} \cup \{H_{k_1,k_p}\}$.*

Proof. First, we need the following hint.

Hint 1. *From the arches that are in $s[i].s_i^{l+1}$ and not in s_i^{l+1}, i.e., H_{k_1,k_h} for all $h : 1 < h \leq p$, only one can be added to $\mathcal{Z}$ since they are all pairwise incompatible.*

Now, let us note that for any position $h : 1 < h < p$, either both or none of H_{k_1,k_h}, H_{k_h,k_p} belong to $\mathcal{Z}$. Indeed, if one of them does not, it is because an incompatible arch is in $\mathcal{Z}$. This incompatible arch must have one extremal variant on the left of k_h and the other on the right and then be incompatible with both H_{k_1,k_h}, H_{k_h,k_p}. Thus, the latter do not belong to $\mathcal{Z}$. In the opposite case, the same reasoning shows that H_{k_1,k_h} and H_{k_h,k_p} belong to $\mathcal{Z}$. This demonstrate the Hint.
Now we prove the lemma. In the first case of the lemma, $H_{k_h,k_p} \in \mathcal{Y}$, then by the previous argument and by $\mathcal{Z}$ maximality, we can set $\mathcal{Z} := \mathcal{Y} \cup \{H_{k_1,k_h}\}$. Then by Hint 1 $H_{k_1,k_p} \notin \mathcal{Z}$. In the second case of the lemma, we have the opposite situation and we can then set $\mathcal{Z} := \mathcal{Y} \cup \{H_{k_1,k_p}\}$. This is correct since none of H_{k_h,k_p} for $h : 1 < h < p$ is in $\mathcal{Y}$ and these arches are the only incompatible arches with H_{k_1,k_p}. □

Generalization of the max stable set problem. For the sake of clarity, we assume that visible arches are mapped to subintervals of the integer subset $[1, n]$ and neglect here the constraint of intervals having distinct end-points. We qualify these subintervals of **real**. As mentioned above, the algorithm of [1] builds the associated overlap graph and computes recursively the max stable set for each subinterval (with the corresponding restricted overlap graph) and then for the whole $[1, n]$. In our problem, to compute the costs of all arch compressions or generations, we need the max stable set for all subintervals of $[1, n]$, i.e., also for subintervals that are not mapped to an arch and that we call **virtual**. In an other work [12], we extend the max stable set problem to the problem of computing the max stable set for each possible subinterval, and report an $\Theta(|V|^2)$ time algorithm to solve it. To save place, we do not present this algorithm here. Applying this algorithm to map s, we can store in an n^2 matrix the max stable set for each subinterval/substring of s. We need the pendant matrix for the arch generations in r. This calculation is made during preprocessing. Storing the max stable set of arches for any substring allows us, first, to compute them only once, and second, to calculate the cost of arches compressions and generations in $O(1)$ during the computation of the DP matrix.

Theorem 4. Algorithm complexity *Our algorithm requires $O(\max(m, n)^4)$ time and $O(\max(m, n)^3)$ space.*

Proof. Algorithm complexity

Matrices of max stable sets : For the map s, we compute a matrix whose entry on the ith line and jth column stores the maximum set of compatible arches of the substring s_j^i. We need the same matrix for the other map, r. There are at most n^2 arches in s and each becomes a node in the overlap graph, such that $|V| := O(n^2)$. Applying our extended algorithm requires time $\Theta(|V|^2)$, i.e., $O(n^4)$. As each matrix element stores a maximal set of compatible visible arches there are at most $O(n)$ arches in each set because of the compatibility definition. An implementation as randomized search trees or balanced search trees requires $O(n)$ memory and allows searches in $O(\log n)$ time. These matrices for maps s and r requires $O(n^3 + m^3)$.

Dynamic programming computation : The matrix $\mathcal{A}$ occupies $O(n \times m)$ memory unit. Any entry depends

on three neighboring entries and at most on all previous entries on the same line and the same column. The computation for the different dependencies can be done in $O(1)$ time units except for phantom arches. Applying Lemma 3 for the phantom arch $s[i].s_i^{l+1}$ requires to test $i-l$ times for set membership in a set in the preprocessing matrix for s. This takes $O((i-l)\log(i-l))$ time. As mentioned in Section 3.2, there are at most $i-1$ arches to consider for the computation of the i-th line of $\mathcal{A}$. Thus, it takes $\sum_{i:=1..n} i^2 \log i = O(n^3 \log n)$ for all lines, and $O(m^3 \log m)$ for all columns. Added up this gives $O(m \times n + n^3 \log n + m^3 \log m) = O(max(m,n)^3 \log(max(m,n)))$.

If p denotes $max(m,n)$, the overall complexities are $O(p^4)$ time and $O(p^3)$ space. □

4. APPLICATION TO THE MINISATELLITE MSY1.

MSY1 is an hypervariable ms locus on the human Y chromosome. Its repeat unit is 25 base pairs long and five different variants have been typed by PCR; they differ from each other by at most 3 substitutions. Known minisatellites are usually GC-rich. In MSY1, the nucleotidic sequences of the variants are 75 to 80% AT-rich and have the ability to form an hairpin structure. This structure may hinder the progression of the DNA polymerase and promote tandem amplification and contraction through slippage during replication. The restricted allele length diversity and the observed modular structures advocate in favor of single-step amplifications and contractions [11]. The model we choose seems to be well-suited to MSY1.

M. Jobling provided us with a set of the MSY1 maps of 609 men, for most of which the haplogroup is known, and with an evolutionary tree of the 27 haplogroups. Most individuals are assigned to a population. We computed with our method all pairwise comparisons between these maps and obtained a 609×609 distance matrix. We used a Neighbor-Joining method, BIONJ [8], to reconstruct an evolutionary tree of these individuals. The percent of explained variance of the tree is 95% and confirm that our distances are tree-like. Our tree does not mirror the structure of the haplogroup tree and individuals of different haplogroups are neighbors in it. This reflects the facts that MSY1 evolves extremely rapidly and is constrained in length. The evolutionary signal of the Y chromosomes evolution is only partly decipherable in the MSY1 maps. At a smaller scale, we can look at the evolution inside an haplogroup. For instance the haplogroup 16 contains chromosomes from the Yakut (8 individuals), Siberian Yakut (5), Finnish (10), Mongolian (23) and others (6) populations. The four main populations are by a majority monophyletic: Yakut and Siberian Yakut together (13/13), Finnish (8/10), Mongolian (20/23). This example shows that comparisons of MSY1 can distinguish between populations inside an haplogroup. This can serve to address micro-evolutionary issues about the Y chromosomes and assign a putative population to an individual of unknown origin. More generally, MSY1 and other ms can play a role in forensic studies and identification. Further results from the MSY1 maps comparisons are under investigation and are not included here.

5. CONCLUSION

In this work, we design the first algorithm for the comparison of minisatellite maps. We utilize this method to compare a large set of human maps from the hypervariable locus MSY1 and reconstruct an evolutionary tree of the individuals. Despite the rapid evolution of MSY1, we could distinguish between populations inside haplogroups. This argues in favor of the validity of our approach to address (micro-)evolutionary issues with minisatellite maps comparisons.

Acknowledgments: We thank M. Jobling for the data, for pointing out the MSY1 case and helping us understanding minisatellite evolution, S. Rahmann, J. Stoye and T. Müller for discussions. S.B. & E.R. are supported by a grant for the French Ministry of Research, the Genopole of Montpellier and the Inter-EPST program for Bioinformatics.

6. REFERENCES

[1] A. Apostolico, M. J. Atallah, and S. E. Hambrusch. New clique and independent set algorithms for circle graphs. *Discrete Applied Mathematics*, pages 1–24, 1992.

[2] J. A. Armour, T. Anttinen, C. A. May, E. E. Vega, A. Sajantila, J. R. Kidd, K. K. Kidd, J. Bertranpetit, S. Paabo, and A. J. Jeffreys. Minisatellite diversity supports a recent african origin for modern humans. *Nat Genet*, 13(2):154–60, 1996.

[3] G. Benson. Sequence Alignment with Tandem Duplication. *J Comput Biol*, 4(3):351–67, 1997.

[4] G. Benson. Tandem Repeats Finder: a Program to Analyze DNA Sequences. *Nucleic Acids Res*, 27(2):573–80, Jan 1999.

[5] G. Benson and L. Dong. Reconstructing the Duplication History of a Tandem Repeat. In *Proceedings of the 7th ISMB*, pages 44–53, Heidelberg, Germany, 1999.

[6] J. Buard and A. J. Jeffreys. Big, bad minisatellites. *Nat Genet*, 15(4):327–8, 1997.

[7] O. Elémento, O. Gascuel, and M.-P. Lefranc. Reconstructing the duplication history of tandemly repeated genes. *Molecular Biology and Evolution*, 2002. in press.

[8] O. Gascuel. BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Mol Biol Evol*, 14(7):685–95, Jul 1997.

[9] A. J. Jeffreys, P. Bois, J. Buard, A. Collick, Y. Dubrova, C. R. Hollies, C. A. May, J. Murray, D. L. Neil, R. Neumann, J. D. Stead, K. Tamaki, and J. Yardley. Spontaneous and induced minisatellite instability. *Electrophoresis*, 18(9):1501–11, 1997.

[10] A. J. Jeffreys, A. MacLeod, K. Tamaki, D. L. Neil, and D. G. Monckton. Minisatellite repeat coding as a digital approach to DNA typing. *Nature*, 354(6350):204–9, 1991.

[11] M. A. Jobling, N. Bouzekri, and P. G. Taylor. Hypervariable digital DNA codes for human paternal lineages: MVR-PCR at the Y-specific minisatellite, MSY1 (DYF155S1). *Hum Mol Genet*, 7(4):643–53, 1998.

[12] E. Rivals. Extension of the Max Stable Set Problem for Overlap Graphs. Technical Report 01-183, LIRMM, 2001.

[13] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits and Macromolecules : the Theory and Practice of Sequence Comparison*. CSLI Publications, second edition, 1999.

[14] M. Tang, M. Waterman, and S. Yooseph. Zinc Finger Gene Clusters and Tandem Gene Duplication. In *Proc. of the 5th RECOMB*, pages 297–304, Montréal, Canada, 2001. ACM Press.

[15] G. Vergnaud and F. Denoeud. Minisatellites: mutability and genome architecture. *Genome Res*, 10(7):899–907, 2000.

Appendix

In this appendix, we include a proof for the distance metric and a proof for the algorithm correctness.

Proof. Distance metric proof We want to show that our alignment cost is a mathematical distance, i.e., that it satisfies i/ the non-negative property, ii/ reflexivity iii/ symmetry and iv/ the triangle inequality. We build our proof on the analysis of metric properties of alignment distances in [13, Chapter 9, p. 307-308]. Some properties of the elementary costs are directly transmitted to the distance between maps: i/ the non-negative property, since all elementary costs are non-negative, ii/ reflexivity, since only the match costs zero and iii/ symmetry, since the cost of all dual operations are the same. It remains to prove iv/ the triangle inequality. Let r, s, t be three maps. Let us denote by $d(.,.)$ our alignment distance between any two maps. We need to show that $d(r,t) \leq d(r,s) + d(s,t)$. The triangle inequality might be violated if we cannot combined the alignment from r to s and from s to t into a cheaper alignment from r to t. This can only be the case if any two successive elementary operations that apply to the same position in the alignment cost less than a single operation. So we need to check this for all possible pairs of elementary operations. For this, slightly more complex notations than the ones used in the rest of the paper are useful. First, the alignment alphabet is $\Sigma \cup \{-\}$ where $-$ is the symbol for the absence of variant. If E is an elementary operation, we denote by $E(a,b)$ the operation E that transforms the **source symbol**, a, into the **destination symbol**, b, with $a, b \in \Sigma \cup \{-\}$. So, if a, b, c are three distinct variants of Σ, we denote by $A(-,a)$ the amplification (it transforms $-$ into a from an adjacent a), by $C(a,-)$ the contraction, by $A_M(-,b,a)$ the amplification+mutation (it amplifies the adjacent a then mutates it into a b, here the third argument is the adjacent variant on the left), by $M_C(b,-,a)$ the mutation+contraction (it mutates a b into an a and contracts the position into the adjacent a, which yields an $-$), by $M(a,b)$ the mutation if $a \neq b$ and the match if $a = b$, by $I(-,a)$ the insertion of an a, and by $D(a,-)$ the deletion of an a. Now, some ordered pairs of two successive operations on the same alignment position are impossible:

- once a position has been inserted, amplified, or amplified + mutated from r to s, it cannot be inserted, amplified, or amplified+mutated again from s to t;
- once a position is present, i.e., after a match or a mutation, the position cannot be inserted, amplified, amplified + mutated again;
- once a position has been removed from r to s, it cannot be removed again from s to t; thus, neither two successive deletions, contractions, or mutation+contractions, nor one of the former followed by a mutation or a match are possible;
- $A(-,a)M_C(a,-,b)$ is impossible since to amplify an a requires an a on the left position and to mutate and contract an a into a b we need a b on the left; for the same reason, $A_M(-,b,a)C(b,-)$ is not allowed.

In Table 1, we show that all remaining possible pairs can be replaced by a single operation or no operation at all. In such pairs, the destination symbol of the first operation must be equal to the source symbol of the second operation. This proof is independent of the H1 hypothesis and thus, we include pairs that are not always optimal, nor possible depending on the neighboring variants. It is straightforward to check that single operation costs less than the pair, which proves the triangle inequality and completes the proof. □

Proof. Algorithm correctness proof
Initialization. The optimal alignments between two empty maps costs 0, between an empty map and a one-variant map costs I or symmetrically D.
Induction hypothesis: We assume that $\forall l, l' : (l \leq i)$ and $(l' \leq j)$ and $(l \neq i$ or $l' \neq j)$, $\mathcal{A}[l, l']$ equals the optimal alignment cost between s_l^1 and $r_{l'}^1$. We prove that the recurrence equation (given p.) correctly computes for $\mathcal{A}[i,j]$. Any alignment between s_i^1 and r_j^1 can be decomposed into an alignment of smaller prefixes plus one ending operation. The recurrence takes the minimum between all such possible alignments.

To complete the proof, we need to prove that for substrings other than arches, it is useless to consider another ordering of elementary operations than the one induced by the DP computation. Assume that when we compute $\mathcal{A}[i,j]$ we need to consider the compression for the substring s_i^l with $l : 1 \leq l < i$. The compression removes the last variant c of s_i^l by either:

- a contraction into the adjacent variant, then the compression cost cannot be better than $\mathcal{A}[i-1,j] + C$,
- or a contraction into a non adjacent variant, then this is the case of an arch compression,
- or a mutation + contraction, then it cannot improve on $\mathcal{A}[i-1,j] + M + C$, since it can be contracted in the previous variant at the same cost.

All these cases are already taken into account in the recurrence. This concludes the proof. □

First	Second	Single	First	Second	Single
$A(-,a)$	$C(a,-)$	none	$A(-,a)$	$D(a,-)$	none
$A(-,a)$	$M(a,b)$	$A_M(-,b,a)$	$A_M(-,b,a)$	$D(b,-)$	none
$A_M(-,b,a)$	$M(b,c)$	$A_M(-,c,a)$	$A_M(-,b,a)$	$M(b,a)$	$A(-,a)$
$A_M(-,b,a)$	$M_C(b,-,a)$	none	$I(-,a)$	$D(a,-)$	none
$I(-,a)$	$C(a,-)$	none	$I(-,a)$	$M_C(a,-,b)$	none
$I(-,a)$	$M(a,b)$	$I(-,b)$	$D(a,-)$	$A(-,a)$	$M(a,a)$
$D(a,-)$	$I(-,b)$	$M(a,b)$	$D(a,-)$	$I(-,a)$	$M(a,a)$
$D(a,-)$	$A_M(-,b,a)$	$M(a,b)$	$C(a,-)$	$A(-,a)$	$M(a,a)$
$C(a,-)$	$I(-,a)$	$M(a,a)$	$C(a,-)$	$I(-,b)$	$M(a,b)$
$C(a,-)$	$A_M(-,b,a)$	$M(a,b)$	$M_C(a,-,b)$	$A(-,b)$	$M(a,b)$
$M_C(a,-,b)$	$I(-,a)$	$M(a,a)$	$M_C(a,-,b)$	$I(-,b)$	$M(a,b)$
$M_C(a,-,b)$	$I(-,c)$	$M(a,c)$	$M_C(a,-,b)$	$A_M(-,c,b)$	$M(a,c)$
$M_C(a,-,b)$	$A_M(-,a,b)$	$M(a,a)$	$M(a,b)$	$C(b,-)$	$M_C(a,-,b)$
$M(a,b)$	$D(b,-)$	$D(a,-)$	$M(a,b)$	$M_C(b,-,a)$	$C(a,-)$
$M(a,b)$	$M_C(b,-,c)$	$M_C(a,-,c)$	$M(a,b)$	$M(b,a)$	$M(a,a)$
$M(a,b)$	$M(b,c)$	$M(a,c)$			

Table 1: This table shows all the possible combinations of pairs of elementary operations.