# Finding Long and Multiple Repeats with Edit Distance

Maria Federico[1], Pierre Peterlongo[2], Nadia Pisanti[3], and Marie-France Sagot[4,5]

[1] Dipartimento di Ingegneria dell'Informazione, University of Modena and Reggio Emilia, Italy
[2] INRIA Rennes – Bretagne Atlantique, EPI Symbiose, Rennes, France
[3] Dipartimento di Informatica, University of Pisa, Italy
[4] Laboratoire de Biométrie et Biologie Evolutive, University of Lyon 1, France
[5] INRIA Rhône-Alpes, France

**Abstract.** We present a tool for detecting long similar fragments that occur two or more times in a set of biological sequences. The problem has interesting applications in the analysis of biological sequences and their correlation, and becomes computationally challenging when a certain non negligible number of insertions, deletions and substitutions are allowed. For this reason exact exhaustive methods are hardly of practical use. In this paper we introduce a tool, FilmRed, that performs this task, and that manages instances whose size and parameters combination cannot be handled by any existing tool. This is achieved by using a filter as a preprocessing step, and by using the information that the filter has gathered also in the successive inference phase. To the best of our knowledge, FilmRed is the first *ab initio* tool that can deal with repeats occurring possibly several times, that have length of hundreds or thousands bases, and whose occurrences may differ in even more than 10 % of their positions in terms of substitutions and indels.

**Keywords:** long repeats, multiple repeats, LTR, transposable elements, edit distance

## 1 Introduction

Genomes are made of an astonishing amount of repeated fragments, in particular in complex organisms as eukaryotes. These repeats are approximate replications of portions of genomes having different ranges and characteristics depending on their origin and function. As for satellites, this can be tandem repeats of few hundred base pairs, segmental duplications of length at least one thousand base pairs and some type of transposons issued from the *copy and paste* process (retrotransposons). For long time, these repeats, mainly occurring in the intergenic regions, were considered as *junk dna*. However, mentality has changed; transposons, for instance, are now believed to have role in immune system [7] and gene regulation [14]. Depending on the species and of the kind of studied repeated element, the average number of occurrences of a repeat, its length and its divergence between occurrences show a large variability. In this paper, we focus on the problem of finding long multiple repeats that may appear dispersed along one whole genome or chromosome, or are common to different genomes/chromosomes. The proposed tool is designed for calling repeats that are multiple (whose occurrences number may be strictly bigger than 2), long (typically of length $\geq 100$ base pairs), and approximate (each pair of occurrences may show substitutions, insertions or deletions in up to 10 to 15 % of their length).

The identification of such repeats, in particular in large and numerous genomes and when the divergence authorized between repeat occurrences is high, is a particularly difficult computational problem. Indeed, exact methods to find multiple repeats

Maria Federico, Pierre Peterlongo, Nadia Pisanti, Marie-France Sagot: Finding Long and Multiple Repeats with Edit Distance, pp. 83–97.
Proceedings of PSC 2011, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-04870-2 

use dynamic programming, leading to a time complexity in $O(n^r)$ with $n$ the average length of sequences and $r$ their number or the number of occurrences of searched repeats. Such a time complexity is unacceptable for practical use unless with toy examples. However, some tools as Speller [19], Smile [12] or Risotto [16] are designed to fin elements that are multiple and possibly spread over large (set of) genomes. However, this tools focus on particular user defined motifs that are indeed repeats but usually small, anchored with mandatory substrings, well conserved and mostly accepting only substitutions between occurrences. On the other hand, there exists a broad range of heuristic based algorithms to find repeats. Some make use of *seeds* for anchoring repeats before the application of dynamic programming and usually perform progressive alignments: combining pairwise alignments beginning with the most similar pair and progressing to the most distantly related. Even if efficient, such tools do not ensure the accuracy and completeness of the found results.

In order to find multiple repeats in reasonable time, it is possible to preprocess the data using a filter. In this framework, a filter is a tool that quickly discards fragments of sequences that may not belong to any searched repeat. After the filtering phase, usually the remaining dataset is much smaller than the original one, allowing the application of a time consuming algorithm. The user may refer to [17] for a state of the art about string filtration.

We propose FilmRed, a combinatorial approach that combines filtering and alignment phases. It is based on the Tuiuiu [15] filter. Tuiuiu is to date the state of the art filter as it filters for multiple repeats while previous filters are designed for repeats having only two occurrences or not taking into account indels. FilmRed (i) uses Tuiuiu as a preprocessing step and (ii) uses pieces of information collected during Tuiuiu runtime to detect, after filtering, real repeats and to find their precise borders and locations, thus finalising the repeats inference task.

## 2 The filter Tuiuiu and preliminary definitions

A *string* is a concatenation of zero or more symbols from an alphabet $\Sigma$. A string $s$ of length $n$ on $\Sigma$ is represented also by $s[0]s[1]\cdots s[n-1]$, where $s[i] \in \Sigma$ for $0 \leq i < n$. The length of $s$ is denoted by $|s|$. We denote by $s[i,j]$ the *substring* $s[i]s[i+1]\cdots s[j]$ of $s$.

We will focus on the problem of finding $(L,r,d)$-*Erepeats*, defined as follow:

**Definition 1 ($(L,r,d)$-Erepeat).** *Given a set $\mathcal{S}$ of one or more input strings, a length $L > 0$, an integer $r \geq 2$, and an edit distance $0 \leq d < L$, we call a $(L,r,d)$-Erepeat a set $\{\omega_1,\ldots,\omega_r\}$ of $r$ words having length in the range $[L-d, L+d]$ occurring in the sequences of $\mathcal{S}$ such that for all $i,j \in [1,r]$, $d_E(\omega_i,\omega_j) \leq d$, where $d_E(\omega,\omega')$ denotes the edit distance between two strings $\omega$ and $\omega'$.*

The definition can be used to model repeats inside one sequence ($|\mathcal{S}| = 1$) or among several sequences ($|\mathcal{S}| > 1$). In the latter case, one can also enforce that the $r$ words occur over $r$ distinct sequences (and thus one needs $|\mathcal{S}| \geq r$). In both cases, should it be $r = 2$, the problem could be solved in quadratic time with dynamic programming just aligning the whole input against itself, but for multiple repeats like those we target, this solution is not feasible. Current exact exhaustive methods can manage input data of very limited size and/or detect repeats with very small values of $d$ (the approximation measure), while again our target is higher as we want

$d$ to be as much as 10 % or 15 % of $L$. On the other hand, heuristics do not guarantee to find all real repeats and, in general, the quality of their result much depends on the absence of noise in the data.

Filters, and in particular lossless filters[1], have been introduced with the goal of speeding up any method (exact or heuristic) by means of a drastic reduction of the input size obtained with the elimination of most of the data that does not contain any repeat. There is a twofold practical impact of filters: exact methods can push (possibly much) further the threshold of their applicability, while heuristics can gain in speed and possibly even obtain results of better quality. In general, a filter is a good filter if it is much faster than the search that it preprocesses (otherwise one would rather directly perform the search), and it is at the same as *selective* as possible, thus leaving the least amount of false positives, which are fragments of the input conserved by the filter and that turn out not to contain a repeat.

The lossless filter TUIUIU is specifically designed to preprocess the inference of $(L, r, d)$-*Erepeats*, and indeed it takes in input the parameters $L$, $r$, and $d$, as well as the input sequence(s). The tool slides a window $w$ of length $L$ along the whole input, checking whether there are at least $r - 1$ other fragments with which $w$ fulfills a specifically designed strong and fast-to-check necessary condition for being at edit distance at most $d$. If this is the case, then the window $w$ is kept, and it is discarded otherwise. The windows taken into account are those starting at each possible position of the sequences (these are roughly as many as the input size $n$), while the fragments for which the condition is checked against $w$ are actually overlapping blocks of size $L + b + d$ occurring every $b$ positions, where $b$ is the smallest power of 2 larger than $d$. This choice, already done by previous filters [4,18] allows to consider only $n/b$ blocks, thus gaining in speed. If a window $w$ fulfills the necessary condition with a block $B$, then we say that $B$ *is a friend* of $w$. The size chosen for blocks ensures that any occurrence of a word of an $(L, 2, d)$-Erepeat is always totally contained in at least one such block (and possibly in two), and hence the filter is still lossless. On the other hand, taking into account blocks rather than all possible fragments of size in $[L - d, L + d]$ starting positions, the selectivity of the filter becomes a bit weaker, as the necessary condition is checked against a block larger than the window, and in particular strictly greater than $L + d$ which is the largest possible size to be at edit distance at most $d$ from a window of size $L$: this can be an additional source of false positives. In other words, the fact that a window has a block as a friend, does not necessarily mean that the block contains a fragment of size $L + d$ that fulfills the condition with $w$, and in this case the block is retained without deserving it.

Summing up, the choice made in TUIUIU has actually been to design a very strong necessary condition for two strings to be at edit distance at most $d$, and to insert this checking in a suitable framework that detects fragments of the input data that fulfill the requirement with respect to at least $r - 1$ others (belonging to distinct input strings when the requirement for the repeat is to occur in $r$ distinct sequences). Doing this, the necessary condition for $(L, 2, d)$-*Erepeats* is actually turned into one for $(L, r, d)$-*Erepeats* for any $r \geq 2$. The fact that TUIUIU keeps a window $w$ that has at least $r - 1$ blocks as friends gives reasonable hope that $w$ and each one of these fragments are at edit distance at most $d$, but there is no indication that these

[1] With *lossless filters*, we refer to methods that filter the data ensuring that no fragments that may contain a similarity is removed.

fragments among themselves are at pairwise distance at most $d$ and also that they will all be kept by the filter. Indeed, any (or both) of the following cases can hold:

1. One or more pair(s) of the friends of $w$ may not even fulfill the necessary condition between themselves. In other words, the window has enough friends but these are not enough friends of each other. If one represents friendships as an edge, this condition can be seen as a guaranteed star shape structure with the window $w$ as a center, while the requirement would actually be a clique.
2. It may turn out that a friend block of a window $w$ is filtered out later during the filtering process because it does not contain any retained window. We call this case *empty block*. In such a case, if finally too many blocks friends of $w$ are empty, then $w$ would be left with less than non empty $r-1$ friend blocks and thus should be disposed.

Both cases can lead to $w$ be a false positive should all these fragments be necessary to $w$ for being part of a $(L, r, d)$-*Erepeat*. For this reason TUIUIU performs an extra check for empty blocks and, above all, multiple passes, to sensibly reduce the amount of such false positives with very little extra time requirement. For more details about TUIUIU and its optimization the reader can refer to [15,6]. In general, when using TUIUIU, we make a double pass as a default choice.

## 3 The algorithm FilmRed

In this section we will describe the pipeline of the algorithm FilmRed (FInding Long Multiple Repeats with Edit Distance) that is designed to exploit informations raised by TUIUIU to find $(L, r, d)$-*Erepeats*. We start with some observations about windows contained in overlapping blocks because the relation between windows and blocks that contain them is critical at some steps of the algorithm, and also because we will eventually merge overlapping blocks that turn out to contain a repeat in order to highlight possibly longer repeats.

### 3.1 Overlapping blocks and blocks merging

In this section we denote with $c$ and $c'$ the starting position of a block. In general we have $0 \leq c \leq n-1$, but all observations and definitions of this section regard cases in which the block contains at least a window and possibly it is not the rightmost block of the input sequence, and hence in such cases $c$ has a more tight upper bound.

**Observation 1** *Given a sequence $S$ and an integer $d$. Let $b$ be the smallest power of 2 larger than $d$. Any word $w$ of length $L$ in $S$ can be totally contained in at most two consecutive blocks of size $L+b+d$. In particular:*

- *words $w = S[j, j+L-1]$ with $j \in [c, c+b-1]$ (and $c \leq n-b$) belong only to the block $B_i = S[c, c+b+d+L-1]$;*
- *words $w = S[k, k+L-1]$ with $k \in [c+b, c+b+d-1]$ (and $c \leq n-b-d-L+1$) belong to the consecutive blocks $B_i = S[c, c+b+d+L-1]$ and $B_{i+1} = S[c+b, c+2b+d+L-1]$.*

**Definition 2.** *Given a sequence $S$, two blocks $B$ and $B'$, starting in $S$ at positions $c$ and $c'$ respectively, are* overlapped *iff $|c'-c| < L-(b+d)$.*

We define the *merging* of two consecutive blocks in the following manner:

**Definition 3 (block merging).** *Given a sequence $S$, let $B_i = S[c, c+b+d+L-1]$ and $B_{i+1} = S[c+b, c+2b+d+L-1]$ be two consecutive blocks in $S$. A larger block $B'_{i+1} = S[c, c+2b+d+L-1]$ of size $L+2b+d$ is obtained* merging *blocks $B_i$ and $B_{i+1}$.*

The definition can be extended in a straightforward way to the merging of $k$ consecutive blocks.

**Definition 4.** *Given a sequence $S$ and $k$ consecutive blocks $B_i, B_{i+1}, \ldots, B_{i+(k-1)}$ of $S$, such that $B_i$ starts at position $c$ in $S$, merging the $k$ blocks we obtain an enlarged block $B'_{i+(k-1)} = S[c, c+kb+d+L-1]$ of size $L+kb+d$.*

### 3.2 Description of the algorithm

In this section we list the steps of the algorithm FilmRed.

**Step 1: filtering step.** The first step is actually to simply apply tuiuiu with double pass, hence including the optimization that allows to discard also some false positives due to empty blocks. With respect to the plain filter introduced in [15,6], in order to collect information which is useful to speed up the successive steps, we extend as follows this first phase. Information about not empty blocks that are friends of each window kept by the filter is stored in the array data structure *friendsOfWindow* whose size is the number of possible windows of length $L$ (that is $n-L$, where $n$ is the length of the input sequences). The entry *friendsOfWindow*$[w]$ of a specific window $w$ contains the list of blocks that are friend of $w$.
At the end of this step, the portion of the input that is left is the one containing kept windows. In this way, a consistent percentage of the initial sequences is removed, and we are left with actual repeats plus some false positives. The possible cases of false positives have actually been described in Section 2 and, summing up, they can be due to one or more of the following reasons:

$FP_{rect}$ : due to choice of checking the filtering condition for windows of size $L$ against blocks of size $L+d+b$.
$FP_{cond}$ : due to the fact that the condition the filter checks is only a necessary condition, but not sufficient.
$FP^*$ : due to the condition being checked between a window and $r-1$ or more blocks (*star* shape) rather than between all such blocks (or actualy windows inside them).

**Step 2: Semiglobal alignment.** In this step, all windows kept by the filter after Step 1 are aligned to all its friend blocks. Only windows that result to have at least $r-1$ other fragments that are at edit distance smaller than $d$ are actually kept. In other words, this step eliminates all $FP_{rect}$ and $FP_{cond}$ false positives. More specifically, this is achieved as follows. For each kept window $w$, a semiglobal alignment between $w$ and $B$ is performed for all blocks $B$ in *friendsOfWindow*$[w]$. The window has length $L$ while the block has length $L+b+d$. We build a rectangular dynamic programming matrix with the window $w$ on rows and the block $B$ on columns. The matrix is initialized with zero on the first row, indels and mismatches cost 1 and matches cost 0. In order to require that $w$ is entirely involved in the alignment, while for $B$ it is

enough to involve a substring, we check the last row: if there is a value lower or equal to $d$, then $B$ contains a repeat of $w$, that is a substring of length in $[L-d, L+d]$ such that its edit distance from $w$ is at most $d$; otherwise, the friendship of $B$ with $w$ was a false positive and $B$ is removed from the list *friendsOfWindow*$[w]$. If with the removal of blocks from *friendsOfWindow*$[w]$ we obtain a list of size lower than $r-1$, then $w$ is no longer a window to be kept, and is thus removed.

Each one of such alignments takes time $L(L+b+d)$, and the number of alignments to be performed depends from the dataset and from the efficiency of the filtering phase, that can only be evaluated experimentally (see Section 4 for experimental results). A theoretical complexity analysis based on the worst case scenario would result in a catastrophic expectation, not at all supported by practical cases. Among the reasons for which this step will actually result feasible there is the fact that we apply a simple optimization with relevant practical impact: when there are consecutive windows to be taken into account, there exists a relationship between the minimum cost of the alignment of a window $w$ against a block $B$, and the minimum cost of the alignment of the successive window $w'$ (that is, the window starting just one position after where $w$ starts) and the same block $B$ (who is likely to belong to *friendsOfWindow*$[w']$ if it did belong to *friendsOfWindow*$[w]$). When considering $w'$ after that $w$ has been processed, we are virtually removing the first row of the alignment between $w$ and $B$, and adding an extra row on the bottom. If we denote with $dist(win, blo)$ the minimum value at the bottom row of the semiglobal alignment of a window $win$ and a block $blo$, then we have that

$$dist(w,B) - 1 \leq dist(w',B) \leq dist(w,B) + 1$$

Therefore, storing for each block $B$, the minimum cost of the alignment with the last aligned window $w$, it is possible to determine lower and upper bounds of the alignment cost between $B$ and the successive window $w'$. As a result, if $dist(w,B) \in [d, d+1]$, then the alignment between $w'$ and $B$ must be computed, but if $dist(w,B) \leq d-1$ (resp. $dist(w,B) > d+1$), then we know for free that $dist(w',B) \leq d$ (resp. $dist(w',B) > d$), and the alignments do not need to be computed.

During this Step, new empty blocks can be introduced: a false positive can be detected and discarded, and hence it may turn out that a block belonging to a list *friendsOfWindow*$[w]$ for some $w$ is actually empty, that is, no window inside it is kept anymore. For this reason, a strategy of removal of empty blocks is performed also during the alignment step. This has the twofold effect of removing on the fly some $FP^*$ and also to spare some alignment computations.

At the end of this step, all false positives $FP_{cond}$ and $FP_{rect}$ have been removed because now for all windows $w$ the *friendsOfWindow*$[w]$ data structure only stores blocks containing at least one substring $x$ of length in $[L-d, L+d]$ whose edit distance with $w$ is $\leq d$. Nevertheless, some $FP^*$ possibly still remain. These will be removed in the next step.

**Step 3: Clique detection among blocks.** At the beginning of this step, we have a set of windows that can be either real repeats or $FP^*$ false positives. For each such window $w$ we do know that in each block belonging to *friendsOfWindow*$[w]$ there is fragment at edit distance no greater than $d$ with $w$, but this is not enough to guarantee $w$ is part of an actual repeat. In order to ensure that, it should be that in any $B_i$ (resp. $B_j$) of such blocks (actually in at least $r-1$ of them) there is a fragment $f_i$ (resp. $f_j$) such that (i) $d_E(f_i, w) \leq d$, and (ii) for all pairs $B_i$ and $B_j$ of

blocks in this set, we have $d_E(f_i, f_j) \leq d$. The existence, for each block, of a fragment that fulfills condition (i) is guaranteed by previous steps, but the point is that the same $f_i$ must fulfill condition (ii) as well. A possible way to see the problem we are about to address, is to represent each window and each fragment $f_i$ as a node of a graph, and to place an edge between two nodes if these are at edit distance at most $d$: in this view, the selection made up to this step ensures that each $w$ is a center of star shaped subgraph that has at least $r-1$ rays, but the actual requirement for this subgraph is now to be a clique. Indeed, a block may contain several (possibly and probably overlapping) fragments that are similar enough to another fragment and to $w$, but the requirement is that a block should be able to pick a single fragment that is similar enough to all other fragments. Somehow, the windows/block asymmetry is what causes this possible shift that may result into a mislocation of the repetition. Also with the goal of overtaking this problem, we relax the constraint over the length $L$ transforming the *friendsOfWindow* data structure in the array of lists *friendsOfBlock* storing for each block $B$ the list of not empty and overlapped blocks that are friends of the windows contained in $B$ and kept after the alignment step.

The construction of the *friendsOfBlock* data structure is performed during the semi-global alignment step contemporarily to the update of the *friendsOfWindow* array. When we find that a window $w_j$ has at least $r-1$ non overlapped friend blocks, if $B_i$ is the block that contains $w_j$, we add the list of friend blocks stored in *friendsOfWindow*$[j]$ in *friendsOfBlock*$[i]$. Note that for the Observation 1 the window $w_j$ can belong to two consecutive blocks $B_i$ and $B_{i+1}$, hence in this case the list of friend blocks stored in *friendsOfWindow*$[j]$ is added to both *friendsOfBlock*$[i]$ and *friendsOfBlock*$[i+1]$.

The *friendsOfBlock* data structure is the adjacency list representation of the graph in which maximal cliques composed of at least $r$ non overlapped friend blocks should be looked for. Blocks composing the found cliques contain occurrences of real multiple repeats having length in $[L-d, L+d]$ and that we can identify and visualize by aligning all the blocks of each clique.

Shifting from windows to blocks at this stage introduces an heuristic step that decreases the complexity of the clique detection task (because the size of the graph is reduced) and maintains the method lossless (*i.e.*, no $(L, r, d)$-*Erepeat* is missed) even though it might prevent the removal of some $FP^*$. We must say that in practice, in all our experiments (that is, all those reported in Section 4, and many more), we have never observed such kind of $FP^*$. Nevertheless, these can theoretically exist.

a. Finding maximal cliques.
   The Bron-Kerbosch [2] algorithm is an algorithm to find maximal cliques in an undirected graph. That is, it lists all subsets of vertices with the two properties that each pair of vertices in one of the listed subsets is connected by an edge, and no listed subset can have any additional vertices added to it while preserving its complete connectivity. We use this algorithm and, namely, the optimised version reported in [2]. This variant of the algorithm involves the selection of a "pivot" vertex for which in [8] two pivot selection strategies are investigated: we tested both on several and distinct types of biological sequences, and we end up choosing as pivot the vertex with largest degree because this strategy always outperforms the one based on random selection.
b. Removing clique redundancy.
   The graph represented by the *friendsOfBlock* array contains overlapped blocks as

friends of a block, therefore the Bron-Kerbosch algorithm performed over such graph finds a set of maximal cliques composed of overlapped blocks, in the sense that there is no clique that is a subset of another one. We have to perform the clique detection considering also overlapped blocks in order to enumerate exhaustively all real repeats. Nevertheless, it is possible that two different cliques in the set actually represent the same repeat. Indeed, for each pair of consecutive entries $i$ and $i+1$ in *friendsOfBlock* corresponding to two consecutive blocks $B_i$ and $B_{i+1}$ that share a window $w_t$ kept after the semi-global alignment step because it has at least $r-1$ non overlapped friend blocks, if $w_t$ is not a $FP^*$ the Bron-Kerbosch algorithm finds two cliques: $C = B_i \cup friendsOfWindow[w_t]$ and $C' = B_{i+1} \cup friendsOfWindow[w_t]$. Aligning blocks of both cliques $C$ and $C'$, the same repeat is found, because the only two different blocks between $C$ and $C'$ contain the same occurrence $w_t$ of the repeat (even if it is possible that blocks $B_i$ and $B_{i+1}$ contain also other overlapped occurrences of the same repeat).

This kind of redundancy in the output is actually avoided storing in *friendsOfBlock*$[i]$ and *friendsOfBlock*$[i+1]$ also blocks $B_{i+1}$ and $B_i$ respectively. In this way the Bron-Kerbosch algorithm finds only the clique $C = B_i, B_{i+1} \cup friendsOfWindow[t]$.

On the other hand, the same type of redundancy now occurs within a single clique, because $B_i$ and $B_{i+1}$ represent the same occurrence of the repeat. Furthermore the same situation may happen also for other blocks in the list of friend blocks of $w_t$ stored in *friendsOfWindow*$[t]$: indeed, for each friend block $B_j$ that contains a kept window $w_k$ belonging also to $B_{j+1}$, both $B_j$ and $B_{j+1}$ are friends of $w_t$, while representing the same occurrence of the repeat.

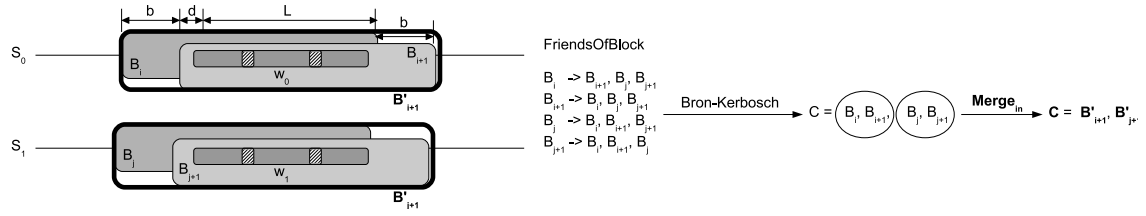


**Figure 1.** *Merge*$_{in}$ operation: given $r = 2$, windows $w_0$ and $w_1$ are two repeat occurrences shared by consecutive blocks $B_i$,$B_{i+1}$ in sequence $S_0$ and $B_j$,$B_{j+1}$ in sequence $S_1$ respectively. The Bron-Kerbosch algorithm finds the clique $C = B_i, B_{i+1}, B_j, B_{j+1}$ in which $B_i$ and $B_{i+1}$ (resp. $B_j$ and $B_{j+1}$) contain the same occurrence. The *Merge*$_{in}$ operation consists in merging consecutive blocks inside the same clique. The white dashed areas possibly contain errors.

In order to remove such kind of redundancy inside a clique we merge consecutive blocks composing it, therefore if we found a clique $C = B_i, B_{i+1}, B_j, B_{j+1}$, we return the clique $C' = B'_{i+1}, B'_{j+1}$. In particular, the merging inside cliques is performed when a new block is added to a candidate clique. Note that the *merge*$_{in}$ operation is applied also for overlapped blocks that are present inside a clique, because they represent overlapped occurrences of the same repeat. We denote the merging of consecutive or overlapped blocks inside a clique as *Merge*$_{in}$. An example is shown in Figure 1.

Of course, more than two consecutive or overlapped blocks may be present in a clique, if they contain overlapped occurrences of the same repeat; in such case only one block that is the union of all consecutive and overlapped blocks is returned as part of the clique, therefore in the alignment we will see only one long occurrence.

Assuming that we perform the merging of consecutive and overlapped blocks inside each found clique, it may happen that the set of found cliques contains subsets of cliques each composed of consecutive blocks:

$$\begin{aligned} C &= B_i, B_j, B_k, B_t \\ C' &= B_{i+1}, B_{j+1}, B_{k+1}, B_{t+1} \\ C'' &= B_{i+2}, B_{j+2}, B_{k+2}, B_{t+2} \\ &\vdots \\ C^n &= B_{i+n}, B_{j+n}, B_{k+n}, B_{t+n} \end{aligned}$$

In this case it is plausible to think that in the input sequences there exists a longer repeat whose occurrences are the concatenation of the occurrences of shorter $n+1$ overlapped repeats with a certain degree of error $d$ represented by cliques $C, C', C'', \ldots, C^n$. When the number of cliques composed of consecutive blocks is huge it means that the user chose a not very accurate value for the length $L$ of repeats to be sought and this produces a little readable output difficult to be managed. To address this problem we decide to merge consecutive blocks contained in the $n+1$ cliques and to return only the clique $C = B'_{i+n}, B'_{j+n}, B'_{k+n}, B'_{t+n}$ representing the longer repeat. We denote the merging of consecutive blocks between

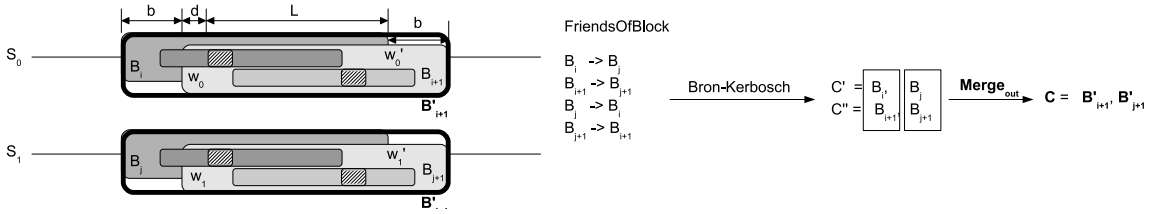


**Figure 2.** *Merge*$_{out}$ operation: given $r = 2$, windows $w_0$ and $w_1$ are two occurrences of the same repeat contained in blocks $B_i$ in sequence $S_0$ and $B_j$ in sequence $S_1$ respectively. Consecutive blocks $B_{i+1}$ and $B_{j+1}$ contain windows $w'_0$ and $w'_1$ which are overlapped to $w_0$ and $w_1$, and are occurrences of another repeat. The Bron-Kerbosch algorithm finds two cliques $C'$ and $C''$ composed of consecutive blocks, but actually in the sequences there exists a long repeat whose occurrences are the concatenation of $w_0$ and $w_1$ in sequence $S_0$, and of $w'_0$ and $w'_1$ in sequence $S_1$. The *Merge*$_{out}$ operation consists in merging consecutive blocks between different cliques. The white dashed areas possibly contain errors.

different cliques as *Merge*$_{out}$. An example is shown in Figure 2.
The two situations of having consecutive and overlapped blocks inside the same cliques and in different cliques may happen simultaneously.
On the contrary, the merging of consecutive blocks contained in two different cliques is not performed if there exist at least two blocks that are not consecutive or overlapped in the two cliques.
Given that:

- as observed in Section 3.1 the enlarged blocks obtained from the merging of $k$ consecutive blocks, have size at most $L + kb + d$;
- each block contains overlapped occurrences of a repeat of length $L$ with an edit distance at most $d$ from each other occurrence of the repeat,

then the enlarged blocks contain occurrences of repeats of length at most $L + kb + d$ with at most $kd$ errors (if areas containing errors in overlapped windows of consecutive blocks are not overlapped, as in Figure 2).

In order to obtain such kind of compressed output, each clique found by the Bron-Kerbosch algorithm (possibly composed of enlarged blocks raised from the merging of consecutive blocks inside the clique) is compared with all previously found cliques and its blocks are merged with blocks of cliques composed of consecutive blocks.

Once the blocks containing the actual repeats have been detected, and that the noise due to redundancy there has been removed, then we are left with the output fulfilling the requirements.

# 4 Experiments and Discussion

## 4.1 Applications of FilmRed

This section shows results of an extensive set of tests performed to validate FilmRed to find $(L, r, d)$-*Erepeats* in biological datasets containing one or more whole genomes or chromosomes of Sunflower, Saccharomyces Cerevisiae, and in the CFTR dataset ([3]) containing, for five different organisms (chicken, cow, human, mouse and tetra), as many ortholog regions of the Cystic Fibrosis Transmembrane conductance Regulator gene. Performances of the different steps of FilmRed will be evaluated in terms of running time. Furthermore, we will also evaluate its selectiveness ability, in terms of amount of data left after the first and the second steps of FilmRed (that is, the filtering step, and the semiglobal alignment steps that removes $FP_{cond}$ and $FP_{rect}$, respectively). The *selection* of these two steps is defined as the ratio between the number of non-removed overlapped substrings of length $L$ and the total number of overlapped substrings of length $L$ present in the input sequences. Formally, the selection of both steps 1) and 2) of FilmRed is given by:

$$sel = \frac{\text{number of words of length L kept by FilmRed step}}{\text{number of words of length L in the input sequences}}.$$

Obviously, given that both phases are lossless, the smaller the selection, the better. On the other hand, for step 3) (that is, the clique detection step and the redundancy removal, respectively), we report the number of output cliques.

**Tests with Sunflower BAC sequences.** A possible application to biological data in which an accurate $(L, r, d)$-*Erepeats* finder can be employed, is that of detecting LTR sequences (LTR is the acronym for Long Terminal Repeats, that are the sequences of about 300 bp length repeated at both ends of a transposable element). In order to check whether this assumption is correct, as a first experiment we applied FilmRed to four different datasets composed of a single BAC sequence of the Sunflower, using length parameters that agree with the expected structure of LTRs ($L = 200, 300$, with $d = 20, 30$, respectively).

Table 1 and Table 2 report results of FilmRed for one of these four datasets denoted as *bacKnapp* and containing 107161 bases, using respectively $r = 2$ and $r = 3$. The results for the other three data sets were practically equivalent to that we report.

Analyzing in detail the performance of the single steps of FilmRed we observe that, as expected, the most time consuming step is the semiglobal alignment between windows and friend blocks (except for the very special case of last line of Table 2 that

| | | Filter | | Semiglobal Align | | Clique detection | | Total |
|---|---|---|---|---|---|---|---|---|
| $L$ | $d$ | time(s) | sel | time(s) | sel | time | #cliques | time(s) |
| 200 | 20 | 0.50 | 12.47 % | 5.13 | 10.32 % | 0.00 | 8 | 5.63 |
| 300 | 30 | 0.49 | 12.12 % | 10.85 | 9.85 % | 0.01 | 5 | 11.34 |

**Table 1.** Performances of the different phases of FilmRed to find $(L, r, d)$-*Erepeats* on the Sunflower *bacKnapp* dataset (107161 bases), with $r = 2$.

we will specifically comment later). However, we are able to perform the alignment task in reasonable time for all parameters (and this holds for all the four datasets), because the pre-processing filtering step sensibly reduces the input size.

The other step with an high theoretical computational complexity is clique detection performed on the graph of friend blocks. However, we observe that, even though the Bron-Kerbosch algorithm applied on an $n$-vertex graph has a time complexity exponential in $n$, the clique detection phase is very fast in all tests, and even more when $r = 3$ instead of $r = 2$, that is when the clique is less trivial, because it is performed on really small graphs of friend blocks, thanks to the filtering of input sequences and the little amount of false positives remaining after the semiglobal alignment step. For what concerns the number of detected cliques, we can deduce that our strategy of compression of the output allows us to obtain a restricted output. Indeed, FilmRed returns very few repeats, especially when $r = 3$. Finally, the last two lines of Table 2 report tests in which the allowed edit distance is pushed quite far (45 edit operations allowed in a 300 bases long repeat means 15 % of the involved bases): no new result raises in this LTR finding task, but we can see that the time performances of FilmRed are good, even if the filters helps much less and takes more time.

In addition, in order to validate our results, we compared repeats found by FilmRed in the Sunflower with the ones found by the signature-based repeat finding tool LTR_Finder [22]. Given that no annotation is available yet, then the output of such a tool is the only result we can compare to. We observed that all the repeats identified by the other tool are found also by FilmRed. The latter, however, returns also further repeats, which are not identified by the former. These results suggest that FilmRed can provide a fast solution to the problem of finding long repeats modeling LTRs.

| | | Filter | | Semiglobal Align | | Clique detection | | Total |
|---|---|---|---|---|---|---|---|---|
| $L$ | $d$ | time(s) | sel | time(s) | sel | time | #cliques | time(s) |
| 200 | 20 | 0.44 | 3.32 % | 3.38 | 1.10 % | 0.00 | 3 | 3.82 |
| 300 | 30 | 0.46 | 3.42 % | 7.36 | 0.98 % | 0.00 | 2 | 7.82 |
| 200 | 25 | 0.59 | 5.66 % | 4.24 | 2.57 % | 0.00 | 3 | 4.83 |
| 300 | 45 | 178.25 | 41.70 % | 35.59 | 3.15 % | 0.00 | 2 | 213.84 |

**Table 2.** Performances of the different phases of FilmRed to find $(L, r, d)$-*Erepeats* on the Sunflower *bacKnapp* dataset (107161 bases), with $r = 3$.

**Tests with Saccharomyces Cerevisiae genomes.** We performed experiments on the dataset *s288c+w303* composed of three whole genomes (16 chromosomes each) of three different strains of *S. cerevisiae*: RefSeq (that is fully annotated in the *Saccharomyces* Genome Database), S288c and W303, for a total of 26392324 bases. The dataset was pre-processed by the regender tool [1] (the reported size is that after regender is applied) in order to remove the resident genome (*i.e.*, the total immotile

DNA), which is equal among all the strains and does not contain mobile elements like transposable elements. The goal of applying FilmRed to this dataset is to detect transposable elements and LTRs that are shared by the three strains, and that could not be detected by means of a traditional global alignment because in general, being part of the most mobile DNA, have lost their colinearity.

| | | Filter | | Semiglobal Align | | Clique detection | | Total |
|---|---|---|---|---|---|---|---|---|
| $L$ | $d$ | time(s) | sel | time(s) | sel | time | #cliques | time(s) |
| 200 | 20 | 29.44 | 0.17 % | 744.48 | 0.09 % | 6.30 | 24 | 780.22 |
| 300 | 30 | 31.68 | 0.16 % | 1473.65 | 0.07 % | 2.13 | 13 | 1507.46 |
| 5000 | 500 | 9.00 | 0 | - | - | - | - | 9.00 |

**Table 3.** Performances of the different phases of FilmRed to find $(L, r, d)$-*Erepeats* on the *s288c+w303* dataset (26392324 bases) of the S. Cerevisiae, with $r = 3$.

Table 3 reports results of tests performed to find $(L, r, d)$-*Erepeats* characterized by the following parameters: $r = 3$, $L = 200, 300, 5000$ with $d = 20, 30, 500$, respectively, in the *s288c+w303* dataset. We chose these parameters based on the peculiar structure of the transposons that can be evinced from the annotation of RefSeq provided in the *Saccharomyces* Genome Database (available at `http://www.yeastgenome.org`): they are long between 5000 and 6000 bases and are delimited by two LTRs of 200-300 bases.

Basically, all the observations we made for the sunflower data set hold here as well, including the fact that our tool is a good candidate to detect LTRs: for this data set an annotation is available, and hence in this case we could really validate our results. In particular, we checked whether the repeats found by FilmRed in this dataset of S. Cerevisiae correspond to real LTRs whose annotation is available in the *Saccharomyces* Genome Database (`http://www.yeastgenome.org`). We found that repeats output using parameters $L = 300$, $d = 30$, $r = 2$ actually correspond to real LTRs, or are part of retrotransposons, or they match with the sequence of putative proteins of unknown function. For example, blocks composing a detected clique contain occurrences of the following annotated LTRs: YCLWdelta3 and YCLWdelta5 in chromosome III, YDRWdelta19 and YDRWdelta28 in chromosome IV, and YLRWdelta14 and YLRWdelta23 in chromosome XII. For longer repeated sequences such as transposons and retrotrasposons, nothing is selected, as expected, probably because the edit distance with 10 % of edit operations is not the right framework to capture transposons' divergence.

### 4.2 Comparison with other tools

As already pointed out, to the best of our knowledge, FilmRed is the first *ab initio* tool that can deal with repeats occurring in possibly more than two sequences, that have length of hundreds or thousands of bases, and whose occurrences may differ in even more than 10 % of their positions in terms of substitutions and indels. For this reason we cannot compare FilmRed with other methods solving the same problem. In this section we report results of experiments performed to compare FilmRed with existing methods for local similarity search. In particular, as the major strength of FilmRed is its capacity to identify repeats in more than two input sequences, we concentrated our attention on existing tools for multiple local sequence alignment. It is important to note, however, that the output provided by FilmRed and the one

provided by multiple local alignment tools are different, because the tasks addressed by the two kinds of tool are different. Indeed, FilmRed returns repeats and their occurrences, while the output of multiple local alignment tools is the alignment of whole input sequences in which we can identify local similarity areas (the repeats) looking at the alignment.

We compared FilmRed with some of the most popular multiple local alignment tools on the CFTR dataset, which is the smallest dataset (5.5 Mbases) composed of more than two sequences that we have studied in our work. Experiments were run on an Intel(R) Quad-core Xeon(R) E5405/2 GHz with 10 GB of RAM.

Table 4 reports results of experiments performed on the CFTR dataset ([3]) composed of 5518041 bases. Experiments were performed using parameters: $L = 100$, $r = 5$ and $d = 7, 12, 14, 15$, with $r = 5$.

| | Filter | | Semiglobal Align | | Clique detection | | Total |
|---|---|---|---|---|---|---|---|
| $d$ | time(s) | sel | time(s) | sel | time | #cliques | time(s) |
| 7 | 64.20 | 0.05 % | 56.56 | 0 | - | - | 120.76 |
| 12 | 1017.51 | 0.01 % | 0.88 | 0 | - | - | 1018.39 |
| 14 | 3772.65 | 0.02 % | 1.41 | 0.001 % | 0.00 | 1 | 3774.06 |
| 15 | 7128.19 | 0.65 % | 740.01 | 0.003 % | 0.01 | 1 | 7868.21 |

**Table 4.** Performances of the different phases of FilmRed to find $(L, r, d)$-*Erepeats* on the CFTR dataset (5518041 bases), with $L = 100$ and $r = 5$.

We can see that for low values of $d$, no repeat is detected, while for larger $d$ there is a repeat that, besides the fact that its occurrences pairwise show 15 % of differences, our tool is fast to find.

| Tool | Class | Result |
|---|---|---|
| MSA [11] | exact | manages sequences at most 50 character long |
| ClustalW [21] | progressive | runs for more than 38 hours |
| TCoffee [13] | progressive | runs out of memory |
| Kalign [10,9] | progressive | runs for more than 28 hours |
| DiAlign [20] | iterative | runs out of memory |
| MUSCLE [5] | iterative | runs out of memory |

**Table 5.** Results of several multiple local sequence alignment tools on the CFTR dataset.

We have tried to search for other tools able to find the same results (that is, for example, the existing repeat of L=100 bases long occurring in all five sequences of the CFTR data set, and with up to 14 % pairwise edit distance between occurrences that is detected by FilmRed) with which we could compare the performances of FilmRed. Table 5 summarizes the results of the comparison. As we can see, none of the tested tools was able to manage in reasonable time and without huge memory usage, inputs as large as the one provided by sequences in the CFTR dataset. On the contrary, as shown in Table 4, FilmRed ends its computation in reasonable time on this dataset with these parameters.

## 5 Conclusion and Perspectives

The problem of finding *long repeats* approximated with edit distance, modelling transposable elements in biological sequences, is computationally challenging when a certain non negligible number of insertions, deletions and substitutions are admitted in

repeat occurrences. For this reason the exhaustive discovery of such repeats might be unfeasible for many instances. We proposed an ab initio method, called FilmRed, which is, to the best of our knowledge, the first tool that can deal with repeats occurring possibly several times, that have length of hundreds or thousands of bases, and whose occurrences may differ in even more than 10 % of their positions in terms of substitutions and indels. This is achieved by using a filter as a preprocessing step in order to discard as many as possible fragments of sequences that are guaranteed not to contain any searched repeat, and using the information gathered during the filtering phase in order to speed up a successive dynamic programming based alignment step performed to infer the repeats. Although, in theory, the current version of FilmRed might return some false positives due to the introduction of a localized heuristic step in the method, we have never observed them in practice. Future work will consist in clearly evaluating the false positive rate and finding a new way for fixing the problem.

## References


1. G. Battaglia, R. Grossi, N. Pisanti, R. Marangoni, and G. Menconi: *Inferring mobile elements in S.Cerevisiae strains*, in Proceedings of International Conference on Bioinformatics Models, Methods and Algorithms (BIOINFORMARTICS), 2011. In press.
2. C. Bron and J. Kerbosch: *Algorithm 457: finding all cliques of an undirected graph.* Communication of ACM, 16(9) 1973, pp. 575–577.
3. M. Brudno et al.: *LAGAN and Multi-LAGAN: Efficient tools for large-scale multiple alignment of genomic DNA.* Genome Research, 13 2003, pp. 721–731.
4. S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron: *q-gram based database searching using a suffix array (QUASAR)*, in ACM Conference on Research in COmputational Molecular Biology (RECOMB 1999), 1999, pp. 77–83.
5. R. C. Edgar: *Muscle: multiple sequence alignment with high accuracy and high throughput.* Nucleic Acids Research, 32 2004, pp. 1792–1797.
6. M. Federico, P. Peterlongo, and N. Pisanti: *An optimized filter for finding multiple repeats in DNA sequences*, in Proceedings of the 8th ACS/IEEE International Conference on COmputer Systems and Applications (AICCSA 2010), IEEE Computer Society Press, 2010, pp. 1–8.
7. J. M. Jones and M. Gellert: *The taming of a transposon: V(D)J recombination and the immune system.* Immunological Reviews, 200(1) 2004, pp. 233–248.
8. I. Koch: *Fundamental study: Enumerating all connected maximal common subgraphs in two graphs.* Theoretical Computer Science, 250 2001, pp. 1–30.
9. T. Lassmann, O. Frings, and E. L. L. Sonnhammer: *Kalign2: high-performance multiple alignment of protein and nucleotide sequences allowing external features.* Nucleic Acid Research, 37(3) 2009, pp. 858–865.
10. T. Lassmann and E. L. Sonnhammer: *Kalign – an accurate and fast multiple sequence alignment algorithm.* BMC Bioinformatics, 6 2005.
11. D. J. Lipman, S. F. Altschul, and J. D. Kececioglu: *A tool for multiple sequence alignment*, in Proceedings of National Acadademy of Sciences, 1989, pp. 4412–4415.
12. L. Marsan and M.-F. Sagot: *Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification.* Journal of Computational Biology, 7(3–4) 2000, pp. 345–362.
13. C. Notredame, D. G. Higgins, and J. Heringa: *T-Coffee: a novel method for fast and accurate multiple sequence alignment.* Journal of Molecular Biology, 302 2000, pp. 205–217.
14. V. Pereira, D. Enard, and A. Eyre-Walker: *The Effect of Trasposable Element Insertions on Gene Expression Evolution in Rodents.* PLoS one, 4(2) 2009, p. e4321.
15. P. Peterlongo, G. T. Sacomoto, A. P. do Lago, N. Pisanti, and M.-F. Sagot: *Lossless filter for multiple repeats with bounded edit distance.* Algorithms for Molecular Biology, 4 2009.

16. N. Pisanti, A. M. Carvalho, L. Marsan, and M.-F. Sagot: *Risotto: Fast extraction of motifs with mismatches*, in LATIN, 2006, pp. 757–768.
17. N. Pisanti, M. Giraud, and P. Peterlongo: *Filters and seeds approaches for fast homology searches in large datasets*, in Algorithms in computational molecular biology, M. Elloumi and A. Y. Zomaya, eds., John Wiley & sons, 2010.
18. K. Rasmussen, J. Stoye, and E. Myers: *Efficient q-gram Filters for finding all epsilon-matches over a given length.* Journal of Computational Biology, 13(2) 2006, pp. 296–308.
19. M.-F. Sagot: *Spelling approximate repeated or common motifs using a suffix tree*, in LATIN, 1998, pp. 374–390.
20. A. R. Subramanian, M. Kauffmann, and B. Morgenstern: *DIALIGN-TX: greedy and progressive approaches for segment-based multiple sequence alignment.* Algorithms for Molecular Biology, 3 2008.
21. J. D. Thompson, D. G. Higgins, and T. J. Gibson: *Clustal W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice.* Nucleic Acids Research, 22 1994, pp. 4673–4680.
22. Z. Xu and H. Wang: *LTR_FINDER: an efficient tool for the prediction of full-length LTR retrotransposons.* Nucleic Acids Research, 35 2007, pp. W265–W268.