STAR: An Algorithm to Search for Tandem Approximate Repeats

Olivier Delgrange, Eric Rivals

BIOINFORMATICS

Vol. 1 no. 1 2004
Pages 1–9

STAR: an algorithm to Search for Tandem Approximate Repeats

Olivier Delgrange[†] *and Eric Rivals*

Université de Mons Hainaut, Service d'Informatique Générale, Avenue du champ de Mars, 6, Mons, 7000, Belgique and LIRMM, CNRS UMR 5509, 161, rue Ada, Montpellier Cedex 5, 34392, France

ABSTRACT

Motivation: Tandem repeats consist in approximate and adjacent repetitions of a DNA motif. Such repeats account for large portions of eukaryotic genomes and have also been found in other life kingdoms. Because of their polymorphism, tandem repeats have proven useful in genome cartography, forensic and population studies, etc. Nevertheless, they are not systematically detected nor annotated in genome projects. Partially because of this lack of data, their evolution is still poorly understood.

Results: In this work, we design an exact algorithm to locate approximate tandem repeats (ATR) of a motif in a DNA sequence. Given a motif and a DNA sequence, our method named STAR, identifies all segments of the sequence that correspond to significant approximate tandem repetitions of the motif. In our model, an Exact Tandem Repeat (ETR) comes from the tandem duplication of the motif and an ATR derives from an ETR by a series of point mutations. An ATR can then be encoded as a number of duplications of the motif together with a list of mutations. Consequently, any sequence that is not an ATR cannot be encoded efficiently by this description, while a true ATR can. Our method uses the Minimum Description Length Criterion to identify which sequence segments are ATR. Our optimization procedure guarantees that STAR finds a combination of ATR that minimizes this criterion.

Availability: for use at http://atgc.lirmm.fr/star.

Contact: rivals@lirmm.fr

Supplementary information: an appendix is available at http://atgc.lirmm.fr/star under Paper and contacts.

INTRODUCTION

Approximate Tandem Repeats (ATR) consist in approximate and adjacent repetitions of a DNA motif. ATR are widespread in eukaryotic genomes and thus important from an evolutionary view-point. Surprisingly, ATR are not annotated consistently in database entries of sequence repositories. Actually, systematic detection of significant ATR in a way that is independent on the motif or on the sequence length is beyond the scope of present methods. Nevertheless, locating ATR represents a relevant issue as polymorphic ATR play an important role in population genetics, forensic medicine, and also in the development of diseases like cancer, epilepsy, and others (Buard & Jeffreys (1997)).

In biology, tandem repeats are classified according to the length of the repeated motif into micro- (below 6 bp), mini-satellites (from 7 to 100 bp) and satellites (above). Computer scientists distinguish between tandem repeats, which contain two copies of the motif, and multiple repeats, with more than two copies.

Related works

Some methods allow to identify in the sequence windows with unusual words composition (Hancock & Armstrong (1994), Claverie & States (1993), Wootton & Federhen (1993)). The sequence of such windows may depart from an ATR. Well known in biology is the RepeatMasker program (see http://repeatmasker.genome.washington.edu), which given a set of repeat sequences (not necessarily tandem repeats) locally aligns the input sequence with any repeat from the set. It is used to mask repeats in a sequence before further analyses. Its ability to find tandem repeats depends on the repeat set. It is not a method devoted to search for ATR.

Among the algorithms that aim at precisely locating tandem repeats, one can identify three classes. In the field of computer science, several fast algorithms deal with searching only two-duplication or exact tandem repeats (among others see Main & Lorentz (1984); Kolpakov & Kucherov (1999); Stoye & Gusfield (2002)). These may be used as filters to point out possible duplicated motifs, but are not appropriate for practical issues in biology. Other methods search for ATR where the copies of the motif may only differ from each other only by substitutions (Kolpakov & Kucherov (2001); Landau *et al.* (2001)). A similar approach (Coward & Drabløs (1998)) discovers periodicities in a sequence without finding the boundaries of the ATR. It performs alignment without indels. These algorithms have good complexities and short computation

[†]also at Université de Lille, LIFL UMR CNRS 8022 , 59655 Villeneuve d'Ascq, France

time in practice, but will miss ATR having undergone any insertion or deletion.

Among the algorithms that locate ATR and authorize substitutions and indels are the works of Rivals *et al.* (1997); Sagot & Myers (1998); Benson (1999). The method in Rivals *et al.* (1997) is limited to small motifs and allows only indels between two occurrences of the motif inside an ATR. Sagot and Myers's method first filters out non repetitive parts of the sequence using statistical properties. On the remaining segments, it enumerates all ATR that fulfill criteria fixed by parameters: minimal number of repeats, range of motif size, maximal number of differences between the repeats and a motif. This is a combinatorially exhaustive approach that identifies several possible motifs and alignments for each ATR, and whose complexity depends exponentially on some parameters. The software Tandem Repeat Finder (Benson (1999)) first searches for significant exact repetitions in the sequence. It then uses these repetitions as anchors and checks if the alignment of the region with an ETR scores above a user-defined threshold. The numbers of ATR reported varies with the threshold and for a given threshold the level of approximation allowed depends on the motif length. So, choosing a threshold for systematic annotation remains an open question.

In this work, we design a new algorithm that detects all significant ATR of a given motif, where significance is assessed using the Minimum Description Length (MDL) criterion. MDL provides an absolute measure of the significance of an ATR independently of the motif. It evaluates how many mutations are allowed in an ATR when compared to an ETR of the best possible length. Our algorithm, STAR, needs no threshold value and optimally locates ATR of any input motif with respect to (w.r.t.) the MDL criterion.

ALGORITHM

Given a sequence s of length n, and a motif m of length p, our algorithm, *STAR* locates all significant ATR of m in s. It uses the MDL criterion to distinguish between significant ATR segments and "random" ones. The MDL criterion is a formal version of the Occam's Razor principle which dates back to the 14th century: "One should not increase, beyond what is necessary, the number of entities required to explain anything". In other words, if there are several hypothetical causes for a phenomenon, the simplest or shortest one is more probably the real one. Following the MDL criterion, lossless data compression was used to analyze genetic sequences in Milosavljevič & Jurka (1993); Grumbach & Tahi (1993); Rivals *et al.* (1997, 1996). The mathematical foundations of such approach lies in *Kolmogorov Complexity Theory* (Li & Vitanyi (1997)).

A DNA sequence s is the description of a DNA molecule. Now consider a lossless compressor that given s outputs s'. s', the compressed version of s, is another complete description of s; indeed, s can be exactly recovered from s'. If s' is shorter than s, if the method effectively achieved compression (which is never guaranteed for any sequence), then s' is a better "description" than s according to the MDL criterion. In fact, a compression method tries to reduce the size of a sequence by exploiting a property $\mathcal{P}$. It re-encodes s relatively to $\mathcal{P}$ and this may compress s or not. The more relevant the property, the better the compression. Here, the property $\mathcal{P}$ of interest is "s contains segments that are significant ATR of m". Kolmogorov Complexity Theory shows that s can only be compressed by such a compressor if it fulfills $\mathcal{P}$ (at least on some segments). On the opposite, if s is not compressed, then it does not satisfy $\mathcal{P}$ (cf. non-randomness tests in (Li & Vitanyi, 1997, Ch. 5, p.377).) Therefore, the compression gain, which measures the size reduction between s and s', is an objective and global evaluation of the relevance of the property $\mathcal{P}$ for s. Indeed, it is a significance measure since the probability of observing a compression gain of d-bits is less than 2^{-d} (Milosavljevič & Jurka (1993)).

In this setup, the efficiency of the compression method is of primary importance. The emphasis of our study lies on the compression optimization. Usually, a compression method makes a blind exploitation of the property everywhere in the sequence. If some segments are really shortened by this way, others are in fact lengthened because they do not fulfill $\mathcal{P}$. For the latter, call them $\mathcal{P}$ incompressible, the original segment sequence would be a shorter, better description. Replacing the encoded version of this segment in s' by this original description would improve the global compression gain. For this, the coding scheme must be adapted to allow switching between the two codes and additional information must be included. We design such a coding scheme and an optimization procedure that allow *STAR* to find the decomposition of s into ATR and $\mathcal{P}$ incompressible segments that is optimal w.r.t. the compression gain. It precisely locates all segments which satisfy our property $\mathcal{P}$. Moreover, it does not need an arbitrary threshold to do so. However, an initial compression method is required to efficiently exploit $\mathcal{P}$ everywhere.

STAR operates in three steps. First, *STAR* aligns the sequence s with a perfect repeat (ETR) of the motif m and obtains an optimal list of mutations that convert this repeat into s and the optimal length for the ETR. This is done by *Wraparound Dynamic Programming (*WDP*)* (Fischetti *et al.* (1993)), which computes the optimal alignments between s and any factor of m^∞ in $O(np)$ time (m^∞ denotes the right infinite repetition of m). From the mutation list, *STAR* evaluates in a second step

the compression gain as if s was a single ATR of m. For a true ATR, one expects that the mutation list is short and that it is more economical to encode the motif, the length of the ETR, and the list of mutations. But often, only some segments of s are true ATR. In this step, called COMPWDP, *STAR* computes the curve of the compression gain over s. This curve is defined at positions just before or just after mutations (which are known from the list). For a true ATR segment of s, it possible to compute its local compression gain by substracting the compression gain at its begin and end positions. A little thinking shows that such a segment does not start or end by a mutation ensuring that the curve is defined at these positions. By the theory, a true ATR segment has a positive compression gain, i.e., the curve increases between its end-points. For segments that are not ATR, coding them (nearly) "litteraly"[‡] takes less bits. To maximize the global compression gain over s, it is thus appropriate to switch between litteral encoding for non-ATR segments and the above-mentioned compression scheme for ATR segments. The third step, TURBOOPTLIFT, achieves this maximization by decomposing s into ATR and non-ATR segments optimally w.r.t. the global compression gain.

The remaining of this section starts with some preliminaries on compression, and presents the two last steps of the algorithm COMPWDP, and TURBOOPTLIFT. For the first step WDP, we refer the reader to Fischetti *et al.* (1993). We finish by the analysis of the time complexity.

Preliminaries

Some concepts about *sequences*, *compression*, *codes* and *ATR* are formally presented here.

A *word* is a finite sequence of letters of an alphabet; it is also called a *sequence*. For a word s, $|s|$ denotes its length and s_i, with $0 < i \leq |s|$ denotes its i^{th} letter. A *factor* of s is made of consecutive letters of s: for $0 < i \leq j \leq |s|$, $s_{i..j}$ denotes the factor $s_i \dots s_j$. The empty word has length 0 and is a factor of every word. The factor $s_{1..i}$ (resp. $s_{|s|-i+1..|s|}$) is the *prefix* (resp. *suffix*) of length i of s. The n^{th} *power* of s, noted s^n, is the word s, concatenated $n-1$ times with itself. In this paper, we consider the *nucleic alphabet* $\mathcal{N} = \{a, c, g, t\}$ and the *binary alphabet* $\mathcal{B} = \{0, 1\}$. A DNA sequence is as a word over $\mathcal{N}$.

A *code* enables us to write items over $\mathcal{B}$. It must be injective to allow a unique deciphering. Let Nuc be the code that maps each nucleotide ($\in \mathcal{N}$) to a two-bit code as follows: $(a, 00), (c, 01), (g, 10), (t, 11)$. Nuc extends to DNA sequences: e.g., the sequence $s := aggcta$ is coded as $\text{Nuc}(s) := 00\,10\,10\,01\,11\,00$.

Given an input sequence s, a *compression method* C computes the *compressed sequence* s' such that the entire sequence s can be reconstructed from s'. Whatever is the *input alphabet*, the *output alphabet* is $\mathcal{B}$. The compression of a sequence is *effective* if it reduces its length. To be able to compare the length of the compressed sequence with the length of the original sequence, the latter must be virtually rewritten with Nuc over $\mathcal{B}$ before comparison. Therefore, the compression of a nucleic sequence s, using method C is effective if $|s'| < |\text{Nuc}(s)| = 2|s|$. The natural way to rewrite a nucleic sequence over $\mathcal{B}$ is to use Nuc because, without any assumption about the nucleic frequencies, it uses the same and minimal number of bits for each nucleotide. The *compression gain* is the number of bits saved by the compression and is given by $|s'| - 2|s|$ for a nucleic sequence s.

A code is *self-delimiting (SD)* if no codeword is a prefix of another codeword. It allows codewords written one after the other in a file to be decoded unambiguously. A compressed sequence is a series of codewords of a SD code. Nuc is SD because all codewords have identical length. It is not true for the usual variable-length binary representation of integers: every codeword (e.g. $10 \equiv 2$) is a prefix of other codewords (e.g. $1000 \equiv 8$).

We denote $FL(x, l)$ the fixed length encoding of integer $x < 2^l$ using l bits (each codeword has enough leading 0s to reach length l). For bounded integers, *fixed-length binary representation* is a SD code for a given codeword length. Encoding unbounded integers requires variable-length SD codes (Apostolico & Fraenkel (1987); Li & Vitanyi (1997)). We use the *Fibonacci code* (Apostolico & Fraenkel (1987)) that represents integers using the Fibonacci numbers as a basis (see Appendix). With $Fibo$ the code length grows logarithmically with the integer, i.e., $|Fibo(x)| \in \theta(\log x)$. Moreover, this code satisfies the ICL property needed by our optimization algorithm (see Section Optimization Algorithm).

Formally, an *Exact Tandem Repeat (ETR)* e of a word m, is a factor of a power of m, i.e., $e := u.m^i.v$ with $i \geq 0$, u (resp. v) a suffix (resp. a prefix) of m. An *Approximate Tandem Repeat (ATR)* of m is defined as an ETR of m which has undergone a small number of mutations. For example, $t\,\underline{att}\,act\,\underline{\ cgt}\,a$ is an ATR of act.

Compression Step (COMPWDP)

COMPWDP computes the curve of the partial compression gain yield by the ATR compression scheme over the sequence, i.e., for some position i in s, the compression gain over $s_{1..i}$. This value is denoted $C_m(i)$. COMPWDP takes as input the output of the WDP, which is the length of the optimal ETR and the list of mutations that transform this ETR into s. After describing the coding scheme, we explain how $C_m(i)$ is computed for all valid positions i.

The ATR coding scheme encodes an ATR of motif m by first writting m in a self delimited format, and then coding the alignment. This scheme is suited for coding true ATR

[‡] I.e., coding their length, and the segment as a sequence (see next Section).

(a) WDP: optimal alignment of sequence s (2nd line) with the motif $m := ttc$ (ETR of ttc on the 1st line).

```
ttCTTcTTc-TT-CTtCTtCTTCTTcTTcTTCt--Tc---TT--
gaCTT-TTaaTTgCTcCTgCTTCTT-TTtTTCggaTaaagTTgg
```

(b) COMPWDP: the compressed sequence s'. The 3-bits codewords $S(b)$ means a substitution of the current letter of m^∞ by b, D means its deletion, and $I(b)$ an insertion of a b before the current position.

$s' = Fibo(2)\text{Nuc}(ttc)FL(2,2)\ Fibo(0)S(g)Fibo(0)S(a)$
$Fibo(3)D\ Fibo(2)S(a)Fibo(0)\ldots I(g)Fibo(0)I(g)$

Fig. 1. Output of WDP (a) and COMPWDP (b) with motif ttc.

whose alignment to an ETR contains mainly identities and can be efficiently encoded by the list of mutations. More precisely, it suffices to write the positions that need to be mutated and the corresponding mutation that allows to recover s from the ETR of m. As the ETR may start at any position in m, the phase of m at the beginning of the alignment needs to be encoded. The coding scheme starts with a preamble followed by the encoding of the list. The preamble includes:

- $Fibo(p-1)$: the motif length (p) minus one since $p > 0$,
- $\text{Nuc}(m)$: the motif m in natural encoding,
- $FL(k, \lceil \log_2 p \rceil)$: the starting phase k in m of the alignment; we know that $0 \leq k < p$.

The alignment is coded as a succession of jumps over segments of identities followed by a mutation. A jump is a position offset coded as an integer. For a given current position, we know which character is at this position in the ETR. It can be shown that there are at most 7 possible mutations (3 insertions, 3 substitutions, one deletion) as we know which character is mutated. So, each mutation can be encoded on a fixed 3-bit codes (since $2^3 = 8$) with one code being unused. Let us denote by q the total number of mutations in the alignment. The list is $l_1, t_1, l_2, t_2, \ldots, l_q, t_q, l_{q+1}$. For $1 \leq j \leq q$, l_j is the jump from the previous mutation position (l_j may equal 0), t_j is the mutation to apply at the current position after the jump. l_{q+1} is the length of the last segment of identities to reach the end of s (it may also equal 0). Each l_j is encoded with $Fibo$ and t_j is coded with the associated 3-bit code. To code the list, one reads the alignment and successively outputs the code for each l_j and t_j until the end. See Fig. 1 for an example of the compression step.

For any $1 \leq j \leq q$, let us map t_j to the last position in s produced by the alignement up to t_j, say i_j, and map l_j to i'_j where $i'_j := i_j$ if t_j is a deletion and to $i'_j := i_j - 1$ otherwise. A prefix of the complete code (preamble+list) up to and including the code of l_j encodes $s_{1..i'_j}$, and the prefix up to and including the code of t_j codes for $s_{1..i_j}$. The indices i_j and i'_j are called **separating positions**. Let $K_m(i)$ denote the code length for the prefix $s_{1..i}$ of s, then $K_m(i)$ is defined only if i is a separating position, since one cannot interpret an uncomplete prefix of the code. The partial compression gain up to a separating position i is the difference between the size of the natural encoding of $s_{1..i}$ and $K_m(i)$, that is $|\text{Nuc}(s_{1..i})| - K_m(i)$. So we have $C_m(i) := 2i - K_m(i)$ by definition of Nuc.

Now, the size in bits of the preamble can be computed in constant time for given m, p and k. $K_m(i)$, as well as $C_m(i)$, can be caculated for all separating positions i by a single pass through the alignment in $O(n)$ time, since the alignement is at most $2n$ long. So, the complexity of COMPWDP is $O(n)$. Moreover, for two separating positions $1 \leq i < j \leq n$, the local compression gain of the segment $s_{i+1..j}$ is given by $C_m(i) - C_m(j)$. Figure 2 shows the partial compression gain on a 1000 bp DNA sequence.

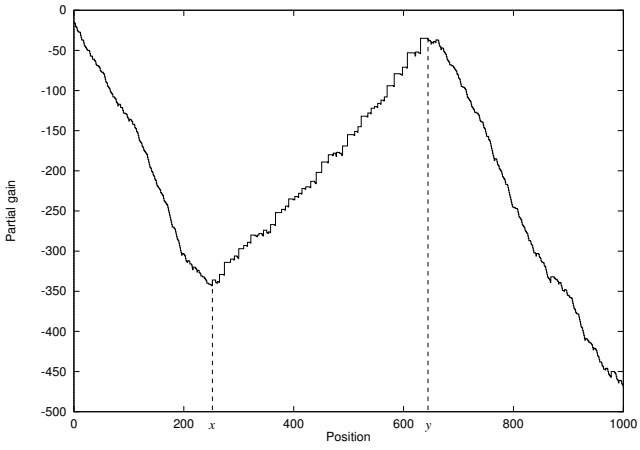

Fig. 2. Compression curve $\mathcal{C}_{\text{TTC}}$ applied to segment $[63700 - 64699]$ of *S. Cerevisiæ* chromosome XI.

Optimization Algorithm (TURBOOPTLIFT)

The compression gain curve yielded by the ATR scheme and computed in COMPWDP contains increasing and decreasing segments. The former are ATR, while the latter are $\mathcal{P}$ incompressible segments. This is illustrated on Fig. 2. As mentioned above, using a more complex coding scheme that allows to alternate between the ATR scheme for increasing segments and the natural scheme for decreasing segments should improve the compression gain. In this section, we first describe the scheme we designed for this purpose and derive the formula for the global compression gain. Then we explain TURBOOPTLIFT whose goal is to choose optimal number and end positions of increasing segments such that the global gain is maximized.

The general coding scheme The general coding scheme is like the ATR scheme, except that it enables interrupting

this scheme by a rupture flag each time one wishes to switch to the natural scheme. The natural scheme is adapted to be Self-Delimited (SD) so that when one reads the compressed sequence, one knows where it ends and where the ATR scheme starts again. Moreover, one has to encode the motif phase of the alignment when the ATR scheme resumes.

The rupture flag, denoted $a_{\mathcal{R}}$, is the 3-bit code that does not correspond to any mutation in the ATR scheme (see page 4), which is a requisite. The natural scheme encodes the sequence segment with Nuc preceded by its length encoded with the $Fibo$ code. The motif phase f is encoded as previously. So, if for some $1 \leq i < j \leq n$ we need to encode $s_{i+1..j}$ in natural encoding, we write $[a_{\mathcal{R}} FL(f, \lceil \log_2 p \rceil) Fibo(j-i) \text{Nuc}(s_{i+1..j})]$ in the compressed sequence.

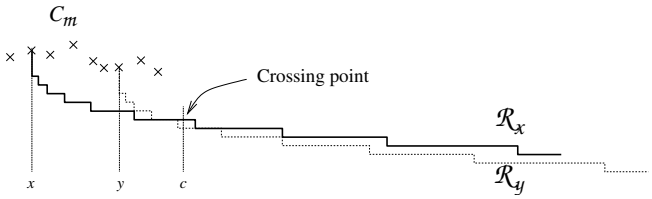

Fig. 3. Two potential rupture curves that cross themselves.

As the length of $\text{Nuc}(s_{i+1..j})$, i.e. $2(j-i)$, is also counted in the original description of s, the compression gain for such a segment is $2(j-i) - |a_{\mathcal{R}} FL(f, \lceil \log_2 p \rceil) Fibo(j-i)| - 2(j-i) = -(3 + \lceil \log_2 p \rceil + |Fibo(j-i)|)$. In this expression, all terms are constant w.r.t. the segment length $(j-i)$, except the term $|Fibo(j-i)|$. The gain is negative: in fact, it is a compression loss. Looking at the compression gain curve, this replaces a decreasing segment of C_m by a piece of the curve shown Fig. 3. The replacement curve, which we call **rupture**, starts with a negative constant (vertical) drop, and continues as a discrete logarithmic curve. This curve depends only on the segment length, but not on its sequence. So, the form of the rupture curve is always the same whichever segment we choose to replace. Changing from the ATR scheme to the natural scheme is termed "applying a rupture", and will usually lift up the segment $[j, n]$ of the original curve C_m as shown Fig. 4. A rupture, which starts necessarily at a separating position, increases the partial compression gain by $\mathcal{C}_m(i) - \mathcal{C}_m(j) - 3 - \lceil \log_2 p \rceil - |Fibo(j-i)|$. Let us denote by $R(l)$ the negative contribution of a rupture of length l to the compression gain: $R(l) := -(3 + \lceil \log_2 p \rceil + |Fibo(l)|)$.

The optimization problem and TURBOOPTLIFT. Given the general coding scheme as described above, we can formally state the optimization problem solved by TURBOOPTLIFT. Assume one wants to apply k non overlapping ruptures in the coding and that the decomposition of s is given by the positions and lengths of the corresponding segments. Let (b_h, d_h) for $1 \leq h \leq k$ be their list, where b_h is the beginning position in s of the h-th rupture minus one and d_h its length. The global compression gain over s is given by: $C_m(n) + \sum_{i=1}^{k} (C_m(b_i) - C_m(b_i + d_i) + R(d_i))$ which follows from the partial compression gain formula written above. The goal of the third step is to find the optimal decomposition of s that maximizes the global gain.

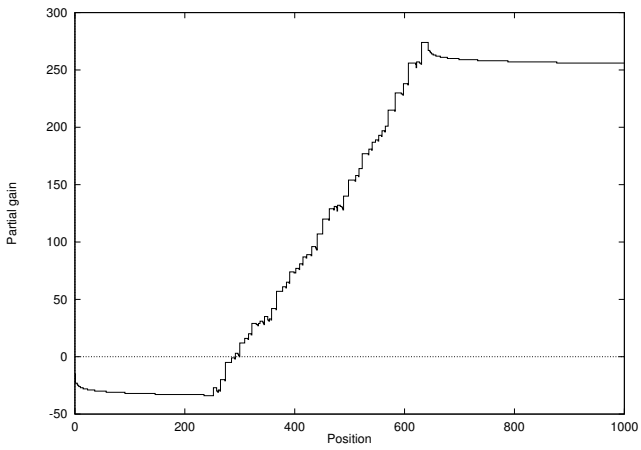

Fig. 4. Compression curve of Fig. 2 after the application of two ruptures.

We show that if the rupture curve satisfies a given property, finding the optimal decomposition is feasible in $O(n \log n)$ time. The problem is complex since a brute force approach would examine an exponential number of potential decompositions. We first exhibit the crucial properties of the rupture curves, before explaining the algorithm and proving its validity. We first need a notation. Let x be a separating position and $\mathcal{R}_x$ denote the potential rupture starting in x. For all $i \geq x$, $G(i)$ is the global compression gain obtained after optimization up to i and $\mathcal{R}_x(i) := G(x) + R(i-x)$ is the value, at position i, of the current compression curve improved by rupture $\mathcal{R}_x$ over $[x, i]$

Rupture curves In fact, we show that fast optimization is possible whenever the rupture length is encoded by a code satisfying the **Increasing Concave and Limited (ICL)** property (see Appendix). It shapes the rupture curve like a discrete, stair-like, logarithmic curve with a step height of one and increasing step's width for successive steps (see Fig. 3).

As there is a constant cost for beginning a rupture, an optimal decomposition will never contain two adjacent ruptures, otherwise it would be better to continue the first until the end of the second. Let $x < y$ be the

Algorithm 1: TURBOOPTLIFT

```
1  I := 0;
2  for i := 1 to n do
3      if i is a separating position then
4          if R_{k_1}(i) > C_m(i) + I then
               // apply R_{k_1} until i;
5              I := R_{k_1}(i) − C_m(i); // update I;
           end
6          else
               // I needs no update;
7              insert rupture R_i in front of L_i;
           end
8          G(i) := C_m(i) + I;
       end
9      // purge L_i, and set L_{i+1} to L_i;
10     for each rupture R_{k+1} of L_i \ {R_{k_1}} in order do
11         if R_{k+1} crosses R_k at pos. i + 1 then
12             remove R_k;
           end
       end
   end
```

starting points of two ruptures. Comparing $G(x)$ and $G(y)$ determines if the curves cross or not, and after which point ones dominates (i.e., is above) the other. If $G(x) > G(y)$ (Fig.3), $\mathcal{R}_x$ crosses $\mathcal{R}_y$ at the smallest $c \geq y$ for which $\mathcal{R}_x(c) > \mathcal{R}_y(c)$, and at the right of c, $\mathcal{R}_x$ always dominates $\mathcal{R}_y$. If $G(x) \leq G(y)$ then $\mathcal{R}_y$ never crosses, and dominates $\mathcal{R}_x$: $\mathcal{R}_y(i) \geq \mathcal{R}_x(i), \forall i \geq y$.

The algorithm The input of TURBOOPTLIFT is the curve C_m. TURBOOPTLIFT scans C_m with the current position i going from position 1 until n and at each iteration optimizes the compression gain up to position i. For this, it maintains the cumulative increase of the compression gain up to the current position, denote it $I(i)$. The improvement is due to the optimal ruptures applied between positions 1 and i. The sum of $C_m(i)$ and $I(i)$ gives $G(i)$. TURBOOPTLIFT computes the value of $G(i)$ for all i and outputs $G(n)$ as the optimal global compression gain. As $I(i)$ is used only in one iteration, it is stored in a single variable I rather than in an array.

TURBOOPTLIFT maintains the list of potential ruptures, denoted L_i; it contains all ruptures $\mathcal{R}_k$, with $k < i$, that could improve the curve at step i or later. This list is sorted by decreasing values of $\mathcal{R}_k(i)$ and by increasing rupture length. This is a total order on the ruptures, which, we will see, is crucial for the list maintenance. $\mathcal{R}_k(i)$ represents the sum of the cumulated increase until position k, plus the increase obtained by applying $\mathcal{R}_k$ until the current position. Thus, the best applicable rupture up to i is necessarily the first in L_i.

TURBOOPTLIFT is outlined in Algorithm 1. Let us denote by k_1 the start position of the first rupture in L_i. Lines 3 to 8 update $G(i)$ and L_i. Line 4 checks whether a rupture needs to be applied until i. Due to the ordering of L_i, this rupture necessarily is $\mathcal{R}_{k_1}$. If $\mathcal{R}_{k_1}$ is applied, the cumulative increase I is correctly updated (line 5). Moreover, the starting rupture $\mathcal{R}_i$ is inserted only if it is not dominated by $\mathcal{R}_{k_1}$, otherwise it would not be useful. All this requires $O(1)$ time. Lines 9 to 12 purge L_i by removing ruptures that are dominated at position $i+1$ by the next rupture in the list. Because L_i is sorted according to the total order, this is done in one left-to-right pass over L_i. Only one test needs to be done for any $\mathcal{R}_{k+1}$ since, if it also dominates $\mathcal{R}_{k-1}$, then $\mathcal{R}_k$ also dominates $\mathcal{R}_{k-1}$ and the latter was removed at the previous iteration. Thus, the inner for loop is done in $O(\#L_i)$ time, and as $\#L_i = O(\log n)$ (see Appendix), the algorithm requires $O(n \log n)$ time in total.

Note that ruptures are considered only at separating positions, but L_i is updated at all positions. Moreover, ruptures are applied only when they improve $G(i)$ and the latter is updated properly (line 8). The purge of L_i removes ruptures that are dominated by another one in L_i, which ensures that only useful potential ruptures are in L_i. Also, $\mathcal{R}_i$ is inserted in L_i only if it can be useful in further stages (see Appendix). This sketches the correctness proof of TURBOOPTLIFT.

The output of the algorithm is the list of all segments not corresponding to ruptures, i.e., that are ATR of m. In the example of Fig. 4 the increasing segment is easy to see. Of course on a larger scale such segments are invisible tiny peaks in the compression curve. Now, we search for all ATR of ttc in the *S. Cerevisiæ* chromosome XI (666448 bp) and the optimized compression curve is shown Fig. 5. Four ruptures are applied and three ATR are located (the three quasi-vertical peaks of the curve). The first is the one found in the 1000-base long window of Fig. 4, meaning that it is not only significant in the small window, but also at the chromosome's scale.

Time Analysis

The overall time complexity needed by *STAR* to find all ATR of a given motif of length p, in a sequence of length n, is $O(np + n \log n)$. Indeed, the time required by WDP is $O(np)$, the one of COMPWDP is $O(n)$, and the time complexity of TURBOOPTLIFT is $O(n \log n)$. This enables *STAR* to analyze large sequences. In practice, all ATR of a motif of 6 bp in a sequence of a million bp can be found in a few seconds on current computers.

RESULTS

Comparison with *Tandem Repeat Finder*.

In this section, we want to assess *STAR*'s ability to find tandem repeats of short motifs (less than 6 bps), i.e., microsatellites. As a complete and biologically verified

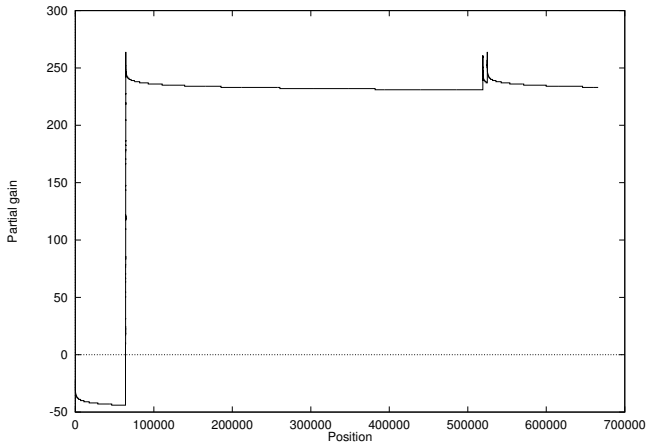

Fig. 5. Optimized curve of $\mathcal{C}^*_{\text{TTC}}$ for *S. cerevisiæ* chromosome XI.

set of microsatellites is unavailable, for a given genome one cannot distinguish the true from the false ATR and evaluate sensitivity and specificity of the method. We choose to compare *STAR* with *Tandem Repeat Finder* (*TRF* Benson (1999)) on the nuclear genome of the baker's yeast, which comprises 16 chromosomes (Goffeau *et al.* (1996)). *TRF* requires several parameters the most influent of which being the alignment costs of matches (M), substitutions (S), indels (D), and the alignment score threshold (T) for an ATR. On the *TRF* website, the default values for these are $M = 2, S = 7, D = 7$ and $T = 50$ (The minimum recommended threshold is $T = 30$). We use these costs as well as another set that penalizes more indels than substitutions $M = 2, S = 6, D = 10$, and threshold values among $30, 36, 40, 50$.

From the point of view of efficiency, *TRF* outperforms *STAR*. To search for all ATR of *a single* motif on a chromosome *STAR* needs as much time as *TRF* when it searches for all ATR of *all* motifs of length less than 6 bp (on the order a second). To detect all microsatellites, *STAR* is run with all possible Lyndon motifs[§] < 6 bp (i.e., 964 motifs) and takes about an hour.

As the two methods differ, *TRF* and *STAR* may not detect an ATR exactly with the same begin and end positions in the sequence. When computing the intersection of their results, we distinguish between ATR of *TRF* that 1/ exactly matches, 2/ is strictly included in, 3/ is overlapped over more than 80, 4/ more than 50 or 5/ less than 50 percent of its length by an ATR of *STAR*. The following table summarizes the average results over all chromosomes for the cost $M = 2, S = 7, D = 7$ and various threshold values. In left to right order, the columns give: the threshold value (`Min Score`), the number of ATR for *TRF* and *STAR* (`# TRF` and `# STAR`), five percentages of ATR found by *TRF* and *STAR* in different categories relatively to *TRF*'s total (`P1` and `P2` for categories 1/ and 2/, `C3, C4, C5` resp. for the cumulated percentage of all categories up to 3, to 4 and to 5), and finally the percentage of ATR found by *TRF* among *STAR*'s ATR.

STAR has no parameter (and does not depend on *TRF* threshold value); it explains why the average number of ATR it finds is constant (column `# STAR`). Indeed, decreasing the threshold from 6 points from 36 to 30 results in an increase of 113% in the number of ATR found by *TRF*. Above the value of 36, *STAR* finds more than 92% of *TRF*'s ATR (columns `C3, C4`). This level increases to more than 98% when the threshold raises to $T = 40$ or more. On the other hand, for all values of T, *TRF* detects at most 55% of *STAR*'s ATR.

Another remark is that more than 33% of ATR are exactly the same in *STAR* and *TRF* output (column `P1`) and more than 22% of *TRF*'s ATR (column `P2`) are included in an ATR of *STAR*, meaning that the latter are longer[¶]. The same table for the alignment costs $M = 2, S = 6, D = 10$ displays similar results (see Appendix), showing that these seem independent of the mutation penalties.

Min Score	Average # TRF	Average # STAR	#(TRF ∩ STAR)/# TRF P1	P2	C3	C4	C5	S
30	162	182	33.06	22.56	63.88	63.94	64.00	55.50
36	76	182	45.00	32.88	92.56	92.62	92.75	38.25
40	54	182	46.06	34.69	98.00	98.12	98.25	28.88
50	26	182	46.06	32.69	99.50	99.75	100.00	13.88

Microsatellites in the genome of *M. jannaschii*

The genome sequence of the archea *Methanococcus jannaschii* is 1664970 bp long (Bult *et al.* (1996)). Searching for microsatellites in the complete sequence we found 41 tandem repeats with lengths in the range $[14, 161]$bp with an average of 43bp. 39 repeats are imperfect and many alignment exhibit insertions, deletions or both. Note that among these repeats, many features segments of perfect repetition larger than 12bp. Four repeats may be classified as composite tandem repeats where several different but related patterns seem to have undergone tandem duplication. For the others, the number of repeats for each pattern size in [1,6] are respectively 1,0,4,5,29 showing a majority of hexanucleotides. The two most frequent patterns are nearly periodic: $aaaaag\|cttttt$, $aaaaat\|attttt$. Moreover, 18 microsatellites are located in intergenic regions, 17 inside a gene and 6 span over a gene boundary.

[§]The set of Lyndon words exclude motifs that are rotation of another (e.g., tac and cta while act remains in the set) or made of the repetition of a shorter motif in that set (e.g., $atatat$ when at is in the set).

[¶]Inclusion happens only twice the other way round.

DISCUSSION

The comparison with *TRF* shows that for a threshold above 36, *STAR* is less efficient, but more sensitive than *TRF*. Indeed, *STAR* finds more than 92% of *TRF*'s ATR, while *TRF* finds at most 38% of *STAR* results. The large variation of the number of *TRF*'s ATR shows how difficult it is to set the threshold. The default value of $T = 50$ suggests that below that it is difficult to separate true ATR from spurious pseudo-repeats. This is not the case with *STAR* since it uses the MDL criterion and the compression gain to select only significant ATR.

Redundancy

For a given imperfect ATR one can find several putative motifs whose exact repetition align well with the ATR (on sometimes slightly different regions). Each one is a possible view or explanation of this ATR (see Landau *et al.* (2001) for a discussion on good definitions of ATR). This means it is difficult to exactly find the boundaries of an ATR and the correct and unique motif. Therefore, *TRF*, Sagot-Myers's method, as well as *STAR*, would give several views of the same ATR segment of a sequence. On one hand, this is an advantage since the biologist may decide which is the best explanation, on the other hand it implies that the output will include some redundancy. Especially, when we search for microsatellites using all possible Lyndon words as motifs (see Appendix), the same region may be seen as several ATR of different motifs.

First ATR with motif $aaag$ from 621220 to 621240 bp.

```
Pat           aaagaaaGa-aagaaaga-aa
Seq    621220 aaagaaaAaGaagaaagaGaa
                  ^ ^       ^
```

Second ATR with motif $aaag$ from 621245 to 621338 bp.

```
Pat           aaag-aaagaaaGaaagaaa-gaaagA-aagaaagaaaGaaAGaaagaaaGaaa-g-a-a
Seq    621245  aaagCaaagaaa-aaagaaaAgaaagGTaagaaagaaa-aaTCaaagaaaAaaaAgGaTa
                  ^       ^     ^^       ^  ^^      ^   ^ ^ ^
Pat           agaaaGaaaga-aaGaaagA-aaGaaagaaag-aaag
Seq    621303 agaaa-aaagaTaaAaaagGTaaAaaagaaagGaaag
                    ^    ^  ^   ^^  ^       ^
```

The overlapping ATR of motif $aaaaag$ from 621220 to 621316 bp.

```
Pat           aaagaaaaagaa-aaagaAaaaGAAaaagAAaaagaaaaaGa-aaaagAaaaAg-aa-aa
Seq    621220 aaagaaaaaagaaGaaagaGaaaCCTaaag-Caaagaaaaa-aGaaaag-aaaGgTaaGaa
                      ^     ^   ^^^    ^^      ^ ^     ^   ^ ^  ^
Pat           agaaaaaG-aaa-aaGaaaaagAaAaagaaaaa-gaAaaa
Seq    621277 agaaaaaTCaaaGaaAaaaaagGaTaagaaaaaAgaTaaa
                  ^^   ^  ^     ^ ^       ^  ^
```

Fig. 6. Example of a composite microsatellite from *M. jannaschii*. The region is seen as two successive ATR with motif $aaag$ and one ATR with motif $aaaaag$. The latter overlaps with both ATR of $aaag$ but ends 22bp before them. We show the three alignments to the corresponding Exact Tandem Repeats of motifs $aaag$ and $aaaaag$.

How does *STAR* handle this redundancy? For each tandem repeat of a given motif, *STAR* computes the consensus motif and checks if it equals the motif given as parameter. This information is output and the user can discard the zones whose consensus is not the motif given as parameter. This is not sufficient to discard all redundancy. Some redundancy remains for instance when the true motif is nearly periodic: an ATR of motif $atatgt$ is often seen as an ATR of motif at and the consensus of the latter is then at, since approximately two-thirds of the motifs are at. The second major case is the one of **compound or composite** microsatellites where several motifs, not necessarily of the same size, have been duplicated and form a single ATR region. An example is given in Figure 6. This can be detected by looking for overlaps of ATR of different motifs. These types of complex microsatellites correspond to both Variable Length or Multi-Periodic Tandem Repeats as defined in Hauth & Joseph (2002), which presents an interesting attempt to handle redundancy. In conclusion, our solution is to let *STAR* output all possible ATR, even if they are redundant. This allows the user to tune up his procedure for curing redundancy.

Influence on the Annotation and Studies of Microsatellites

Among the microsatellites we detected, none are annotated in the corresponding EMBL entry of the genome. This is generally the case for archeal and bacterial genomes, although some microsatellites are known to be functionally active in controlling gene expression. This is presumably due to the difficulty of defining a standard and consistent way of annotating such repeats. Which degree of imperfectness should be allowed in a repeat? Should it depend on the pattern size? Our method assesses the significance of approximate tandem repeats using the same information criterion regardless of the pattern length. Thus, it provides a consistent solution for microsatellites annotation. Moreover, the analysis of *M. jannaschii* summarized above shows that even long tandem repeats occur in such a genome. Their systematic annotation with a guaranteed software like *STAR* may help investigating their roles in the genome structure. When located inside a gene, it may help the functional annotation, while intergenic repeats could point out sequences involved in gene regulation.

Several biological studies of microsatellites in whole genomes use simple protocols to search for microsatellites (Field & Wills (1998); Cox & Mirkin (1997); Nadir *et al.* (1996); Katti *et al.* (2001)). Some collect data on exact tandem repeats of short motifs (Field & Wills (1998); Cox & Mirkin (1997); Nadir *et al.* (1996)). Others have designed specific ad-hoc software to detect slightly mutated tandem repeats that satisfies some arbitrary criterion (e.g. Katti *et al.* (2001) search for ATR with less than 1 substitution in 20 base pairs). On the basis of these results, the authors consider evolutionary issues or other biological questions.

For instance, the evolutionary origin of microsatellites in eight genomes, among which *M. jannaschii*, is investigated in Field & Wills (1998) . For all microsatellite patterns (between 1 and 6bp), the authors report the numbers of exact tandem repeats by length. For *M. jannaschii*, they do not find any repeat larger than 10 bp except a stretch of 24-G. It is clear that their unguaranteed software probably missed numerous perfect repeats segments of size $\geq$ 12 bp that are included in imperfect longer repeats as shown in our results. This is also the case for *A. fulgidus* (data not shown). Moreover, looking only at perfect repeats rises other problems. First, a long imperfect repeat may be counted say twice if it contains two perfect segments. Second, the core data of the study is incomplete since, even in a archea like *M. jannaschii*, our result show that less than 5% of the microsatellites are perfect (2 out of 41 for length $\geq$ 14bp). The use of a program allowing point mutations in the repeats, like *STAR*, should improve future investigations of microsatellites structural and evolutionary characteristics.

Acknowledgments: E.R. is supported by a Bioinformatics Inter-EPST project, Montpellier Genopole, Genoplante, Specific Action #185 of CNRS-STIC, a regional BioSTIC project. O.D. and E.R. thanks M. Dauchet for his helpful suggestions, and F. Lethiec for the new web interface.

REFERENCES

Apostolico, A. & Fraenkel, A. (1987). Robust transmission of unbounded strings using Fibonacci representations. *IEEE Trans. Inform. Theory*, **33**, 238–245.

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

Buard, J. & Jeffreys, A. J. (1997). Big, bad minisatellites. *Nat Genet*, **15**, 327–8.

Bult, C. J., White, O., Olsen, G. J., Zhou, L., Fleischmann, R. D., Sutton, G. G., Blake, J. A., FitzGerald, L. M., Clayton, R. A., Gocayne, J. D., Kerlavage, A. R., Dougherty, B. A., Tomb, J. F., Adams, M. D., Reich, C. I., Overbeek, R., Kirkness, E. F., Weinstock, K. G., Merrick, J. M., Glodek, A., Scott, J. L., Geoghagen, N. S. & Venter, J. C. (1996). Complete genome sequence of the methanogenic archaeon, Methanococcus jannaschii. *Science*, **273**, 1058–73.

Claverie, J.-M. & States, D. J. (1993). Information Enhancement Methods for Large Scale Sequence Analysis. *Comp. Chem.*, **17**, 191–201.

Coward, E. & Drabløs, F. (1998). Detecting periodic patterns in biological sequences. *Bioinformatics*, **14**, 498–507.

Cox, R. & Mirkin, S. M. (1997). Characteristic enrichment of DNA repeats in different genomes. *Proc Natl Acad Sci U S A*, **94**, 5237–42.

Field, D. & Wills, C. (1998). Abundant microsatellite polymorphism in saccharomyces cerevisiae, and the different distributions of microsatellites in eight prokaryotes and s. cerevisiae, result from strong mutation pressures and a variety of selective forces. *Proc Natl Acad Sci U S A*, **95**, 1647–52.

Fischetti, V. A., Landau, G. M., Sellers, P. H. & Schmidt, J. P. (1993). Identifying periodic occurrences of a template with applications to protein structure. *Inf Proc Letters*, **45**, 11–18.

Goffeau, A., Barrell, B. G., Bussey, H., Davis, R. W., Dujon, B., Feldmann, H., Galibert, F., Hoheisel, J. D., Jacq, C., Johnston, M., Louis, E. J., Mewes, H. W., Murakami, Y., Philippsen, P., Tettelin, H. & Oliver, S. G. (1996). Life with 6000 genes. *Science*, **274**, 546, 563–7.

Grumbach, S. & Tahi, F. (1993). Compression of DNA Sequences. In *Data Compression Conf.*. IEEE Comp. Soc. Press, pp. 340–350.

Hancock, J. M. & Armstrong, J. S. (1994). Simple34: an improved and enhanced implementation for vax and sun computers of the simple algorithm for analysis of clustered repetitive motifs in nucleotide sequences. *CABIOS*, **10**, 67–70.

Hauth, A. & Joseph, D. A. (2002). Beyond Tandem Repeats: Complex Pattern Structures and Distant Regions of Similarity. *Bioinformatics*, **18**, S31–S37.

Katti, M. V., Ranjekar, P. K. & Gupta, V. S. (2001). Differential distribution of simple sequence repeats in eukaryotic genome sequences. *Mol Biol Evol*, **18**, 1161–7.

Kolpakov, R. & Kucherov, G. (1999). Finding maximal repetitions in a word in linear time. In *40th FOCS*. IEEE Comp. Soc. Press, pp. 596–604.

Kolpakov, R. & Kucherov, G. (2001). Finding approximate repetitions under Hamming distance. In *ESA: Annual European Symposium on Algorithms*, volume 2161 of *Lecture Notes in Computer Science*. Springer, pp. 170–181.

Landau, G. M., Schmidt, J. P. & Sokol, D. (2001). An algorithm for approximate tandem repeats. *J Comp Biol*, **8**, 1–18.

Li, M. & Vitanyi, P. M. (1997). Introduction to Kolmogorov Complexity and Its Applications. Springer-Verlag.

Main, M. & Lorentz, R. (1984). An $o(n \log(n))$ algorithm for finding all repetitions in a string. *J of Algorithms*, **5**, 422–432.

Milosavljevič, A. & Jurka, J. (1993). Discovering Simple DNA Sequences by the Algorithmic Significance Method. *CABIOS*, **9**, 407–411.

Nadir, E., Margalit, H., Gallily, T. & Ben-Sasson, S. A. (1996). Microsatellite spreading in the human genome: evolutionary mechanisms and structural implications. *Proc Natl Acad Sci U S A*, **93**, 6470–5.

Rivals, E., Dauchet, M., Delahaye, J.-P. & Delgrange, O. (1996). Compression and genetic sequences analysis. *Biochimie*, **78**, 315–322.

Rivals, E., Delgrange, O., Dauchet, J.-P. D. M., Delorme, M.-O., Hénaut, A. & Ollivier, E. (1997). Detection of significant patterns by compression algorithms: the case of Approximate Tandem Repeats in DNA sequences. *CABIOS*, **13**, 131–136.

Sagot, M. F. & Myers, E. W. (1998). Identifying satellites and periodic repetitions in biological sequences. *J Comp Biol*, **5**, 539–53.

Stoye, J. & Gusfield, D. (2002). Simple and Flexible Detection of Contiguous Repeats Using a Suffix Tree. *Theo Comp Sci*, **27**, 843–856.

Wootton, J. & Federhen, S. (1993). Statistics of local complexity in amino acid sequences and sequence database. *Comp. Chem.*, **17**, 149–163.