Application and Algorithm: Maximal Motif Discovery for Biological Data in a Sliding Window

Miznah H. Alshammary(✉), Costas S. Iliopoulos, Manal Mohamed, and Fatima Vayani

Department of Informatics, King's College London, London, UK
{miznah.alshammary, c.iliopoulos, manal.mohamed, fatima.vayani}@kcl.ac.uk

Abstract. Since the discovery of motifs in molecular sequences for real genomic data, research into this phenomenon has attracted increased attention. Motifs are relatively short sequences that are biologically significant. This paper utilises the bioinformatics application of the algorithm outlined in [5], testing it using real genomic data from large sequences. It intends to implement bioinformatics application for real genomic data, in order to discover interesting regions for all maximal motifs, in a sliding window of length ℓ, on a sequence x of length n.

Keywords: Motif discovery · Sequence motifs · Bioinformatics application

1 Introduction

Alongside the advancement of next-generation sequencing technology, there has been an increase in the production of genomic data requiring de novo assembly and analyses, one such analysis is motif discovery [1, 8–12]. Motifs, whilst biologically significant, are relatively short sequences. Examples of motifs include protein-binding sites, such as transcription factor recognition sites [4]. This paper captures the bioinformatics application of the maximal motif discovery to solve real genomic data problems. Using the algorithms in [5], which differ significantly from the well-established (ℓ, d)-motif search problem, this paper seeks to find all ℓ length motifs from a given collection of sequences that occur in at least k sequences, where each occurrence of the motif can contain up to d mismatches [13].

The restricted length of the motif is one limitation of (ℓ, d)-motif search approaches. in fact, a longer or shorter motif could be more significant. Thus, this paper focuses on the more general problem of maximal motif discovery. As a maximal motif m˜ d, k is not determined by a given length, its significance is based on its number of occurrences compared to its substrings. A motif is considered maximal, as it cannot be extended to the left or right without its number of occurrences being reduced.

As importance is placed on the number of occurrences, the first parameter k sets a minimum threshold for the number of occurrences of a reported maximal motif. A motif m˜ d, k, occurs in the ℓ-length window ending at position i in a given string X,

© IFIP International Federation for Information Processing 2020
Published by Springer Nature Switzerland AG 2020
I. Maglogiannis et al. (Eds.): AIAI 2020 Workshops, IFIP AICT 585, pp. 213–224, 2020.
https://doi.org/10.1007/978-3-030-49190-1_19

each of which must occur at least k times in the window and contain at most d ‘don’t care symbols’.

The first parameter k sets a minimum threshold for occurrences of maximal motifs. The second parameter d is more restrictive: mismatches occur in up to d specific positions in the motif, known as ‘don’t care letters’ and denoted by ◊. Therefore, a motif is also maximal, as its ‘don’t care letters’ cannot be specialised without reducing its number of occurrences. For example, given the sequence ACGTTATGTT and d = 1, it should be concluded that the significant motif is A◊GTT rather than, for instance, GTT, both of which have the same number of occurrences.

However, this important observation would be missed if the restriction of $\ell = 3$. Notably, a purely de novo approach is used, with only one sequence needed as input.

Furthermore, Grossi et al. [3] proposed that the most current combinatorial solution for maximal motif discovery is a data structure termed a motif trie. A motif trie [3] represents all prefixes, suffixes and occurrence positions of each maximal motif m˜ d, k in the set Md, k of maximal motifs.

This research presents an output-sensitive algorithm with a time complexity of $\mathrm{O}(\mathrm{nd} + \mathrm{d3} \cdot \sum \mathrm{m}^{\sim} \mathrm{d}, \mathrm{k} \in \mathrm{Md}, \mathrm{k}\ |\mathrm{occ}(\mathrm{m}^{\sim}\ \mathrm{d}, \mathrm{k})|)$, where occ(m˜ d, k) is the set of occurrences of m˜ d, k, assuming the input sequence of length n is built on a constant-sized alphabet. Through further research, Iliopoulos et al. [5] proposed the first online algorithm to find occurrences of all maximal motifs in a sliding window in time, where w is the size of the machine word $\mathrm{O}(ndl + d\lceil \frac{l}{w} \rceil \cdot \sum_{i=l}^{n-1} |\mathrm{DIFF}_{\mathrm{i}}^{\mathrm{i}}1|)$ and DIFFii−1 is the symmetric difference of the set of occurrences of maximal motifs at x [i−ℓ..i−1] and at x [i−ℓ+1..i]. The space complexity of the algorithm presented in this paper is O(ℓ2). This results suggests an improvement in the time required to solve the same problem using the motif trie [2]; this would instead be $\mathrm{O}(ndl + d^3 \cdot \sum_{i=l, \tilde{m}_{d,k} \in \mathcal{M}_{i,d,k}}^{n-1} |\mathrm{occ}(\tilde{\mathrm{m}}\ \mathrm{d}, \mathrm{k})|)$. This presents a significant improvement, as a single occurrence of a maximal motif would be reported O(ℓ2). times when using the latter approach. Therefore, the proposed algorithm results in an acceleration of O(d2w) per occurrence of a maximal motif.

Implementing bioinformatics applications was the main motivation for creating a dynamic structure, in order to facilitate a sliding window on the input sequence. Specifically, this endows the additional ability to discover interesting ℓ-length regions of the sequence. This is particularly useful in various forms of bioinformatics, including the prediction of the origin of chromosomal replication (OriC) [6]. The length of OriC in model bacterial species ranges from 120 to 300 bp; for example, in E. coli, it is 240 bp E. coli [7].

Additionally, motifs that occur within OriC, such as DnaA boxes, show that d and k are small constants (for example, d = 2 and k = 4) in practice [2]. Before presenting the results of this paper, the following parameters need to be defined: m˜ d, k occurs in the

ℓ-length window ending at position i in a given string X, each of which must occur at least k times in the window and contain at most d 'don't care symbols'. These definitions are used throughout the rest of the paper. The organisation of the rest of this paper is as follows. In Sect. 2, we present preliminaries in detail. This is followed by Sect. 3 where we summarise our contribution and the experimental result to this problem. Finally, we briefly conclude and state the future work in Sect. 4.

2 Preliminaries

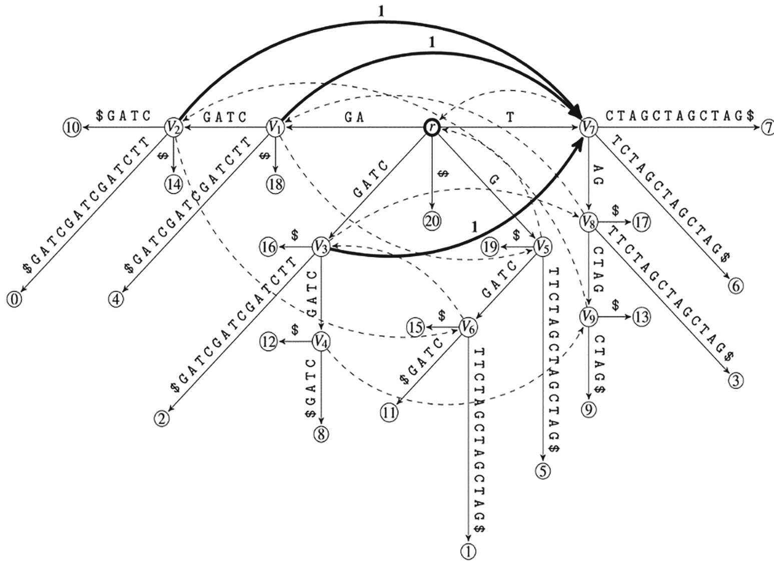
The proposed existing algorithm works online; to use this effectively, one must add a string of letters $(/\in \Sigma)$ to the start of x, whilst also appending a unique letter $(/\in x)$ to the end of x to ensure that the motif graph (augmented suffix tree Tx) of the final window is clear.

2.1 Maximal Motif Discovery Algorithm

In order to redefine and extend the original definition of the motif graph, each internal node V of the suffix tree ST is outlined with the following:

- For an internal node *V*, an integer variable $|occ(V)|$, which holds the number of occurrences of *V* in the window of ℓ-length on *X* and is Seed(*V*), is now defined relative to the window, not the whole string, Seed(*V*) is a Boolean variable, which is TRUE if node *V* is a seed in the window, and FALSE otherwise.
- Each node *V* where Seed(*V*) = TRUE is further augmented with the following:
- The Boolean variable Motif (*V*) is TRUE if node *V* represents a singleton motif in the window, and FALSE otherwise.
- A bit-vector B(*V*) of total size bits shows the occurrence positions of *V* in the window. In order to maintain B(*V*) efficaciously, an integer variable pivot(*V*) is introduced, which represents an anchor so that B(*V*) is only updated when an occurrence of *V* is added or deleted, rather than in every step i of the algorithm.

Example 1. Given the string x = **AGCTAGTTCTAGCTAGCTAG$**, the set of seeds is {V1 = AG, V2 = AGCTAG, V3 = CTAG, V4 = CTAGCTAG, V7 = T}. For d=1 and k=2, we find the following set M1, 2 of motifs from the motif graph shown in Fig. 1.



$\mathscr{B}_{V_1}$	1	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0
$\mathscr{B}_{V_2}$	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
$\mathscr{B}_{V_3}$	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0
$\mathscr{B}_{V_4}$	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0
$\mathscr{B}_{V_7}$	0	0	0	1	0	0	1	1	0	1	0	0	0	1	0	0	0	1	0	0	0

Fig. 1. The motif graph for the suffix tree of the string x = **AGCTAGTTCTAGCTAGCTAG$,** given d = 1 and k = 2. Each leaf node has been labelled with the index i of the suffix X[i..n−1] that it represents, where $i \in [0, n)$. Suffix links are shown as dashed directed edges. The set of all internal nodes,{v1,…,v9}, represents the right-maximal repeated factors of x. As |occ(v5)| = 5 = |occ(v1)|, v5 is not a seed; as |occ(v1)| = 5 ≠ |occ(v8)| = 4, v1 is a seed, and so on. Thus, the set of all seed nodes is {v1, v2, v3, v4, v7}. Motif edges and their labels are bold, All other explicit nodes are labelled V and root node r is outlined in bold. Suffix links are shown as dashed directed edges.

$\tilde{M}_{1,2}$	AG	$AG\diamondsuit T$	$AGCTAG$	$AGCTAG\diamondsuit T$	$CTAG$	$CTAG\diamondsuit T$	$CTAGCTAG$
$\lvert occ(\tilde{M}_{1,2}) \rvert$	5	4	3	2	4	3	2

2.2 Adding a Letter from the Right

When a letter x[i]=α has been added to the right of the window, the following cases are checked in order, only in the case of |occ(α)| ≥ k in the window.

If α now extends at least k occurrences of some motif $M^{\sim} \in$Mi, d, k, it becomes the suffix of a new motif $M^{\sim\prime}$ = M◇d′, α, which happens in the window at least k times, where $d' \in [0, d]$. In this case, the new motif M˜ ′ is added to Mi, d, k. If the number of occurrences of M˜ is equal to M˜ ′, M˜ is deleted from Mi, d, k as it is no longer maximal. The letter α can be added to Mi, d, k as a singleton motif if and only if it is not already in Mi, d, k and one of the following is true:

- No motifs have been added to M*i, d, k*
- One motif *M*˜ ′ has been added to M*i, d, k* and|occc(α)|>|occ(*M*˜ ′)|;
- Two or more motifs have been added to M*i, d, k*

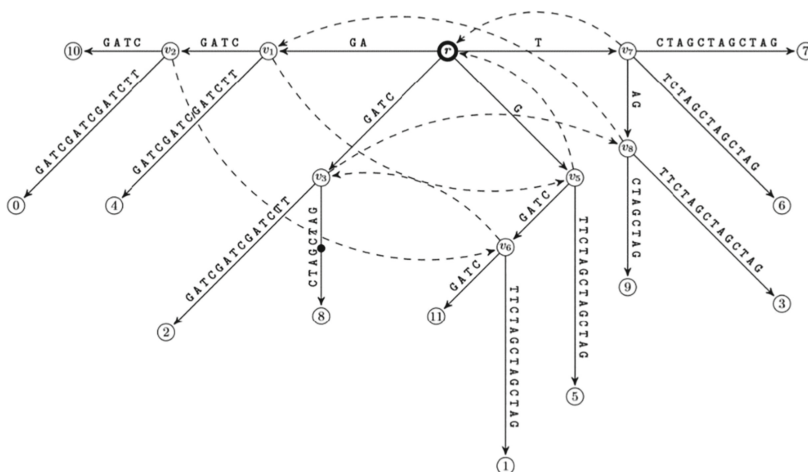


Fig. 2. The implicit suffix tree of the string x = AGCTAGTTCTAGCTAGCTAG. Each leaf node has been labelled with the index of the suffix j that it represents, for all suffixes $j \in [0, 12)$. Observe that, the most recent leaf to be added is suffix j−1 = 11. The active point, $A = \langle v3, 4, 15 \rangle$, The active point is a black dot on the path between nodesV3 and node 8. The longest repeated suffix x[12..19] = CTAGCTAG is the path from r to A. and root node r is outlined in bold. Suffix links are shown as dashed directed edges.

Example 2. Observe that, for example, suffixes 10 and 11 were added as leaf nodes when letter X[16] = C was added.

2.3 Deleting a Letter from the Left

When the leftmost letter α of the window has been deleted, every motif $M^{\sim\prime} = \alpha M^{\sim} i, d, k$ must be deleted if now $occ|M^{\sim\prime}| < k$ in the window. After this possible deletion, the following cases might be considered:

- If $M^{\sim} = \Sigma$, and thus $M^{\sim\prime} = \alpha$, then nothing more is done.
- If $M^{\sim} \neq \Sigma$, then $M^{\sim}$ is added to M*i, d, k*, if and only if $M^{\sim} (\notin Mi, d, k)$ and it is not a prefix of a motif $M^{\sim\prime\prime} \in Mi, d, k$ such that $|occ|M^{\sim\prime\prime}| = |occ|M^{\sim}|$. Finally, the results are printed *Sx, where Sx[i]=|Mi, d, k| and* $\ell \leq i < n$.

Example 3. Considering Fig. 2, the deletion of the leftmost letter (A) would result in the deletion of leaf node 0 which represents the longest suffix. This would cause the subsequent deletion of node V2 as it would have one remaining child, the edges from V1 to V2 and V2 to leaf node 10 would be merged.

3 Experimental Results and Contribution

3.1 Contribution

The patterns that are the focus of this paper are commonly found in molecular structures and, therefore, are biologically important.

A traditional desktop-based programming environment was used to display the coding program of the bioinformatics application using Java programming language. The experiment required an Intel Pentium CPU4415U@ 2.30 GHz processor, 4.00 GB of RAM, a 64-bit Operating System, Windows 10 and MS.Net Framework 4.5.

This experiment sought to find interesting regions within genomes by finding the maximal motif [5] in the proposed algorithm that works in an online manner. Firstly, the main computational challenge in reporting motifs in a sliding window is the maintaining of the left- and rightmost seeds, which is twofold: verifying their maximality (nodes) and updating their relationship with neighbouring seeds (edges). Such changes to the motif graph identify and, therefore, efficiently update only the subset of motifs occurring at both ends of the window. The following describes the effect on Mi, d, k and the motif graph when adding a letter to the right of the window, and deleting a letter from the left, which simulates the sliding window on x.

This paper has sought to further discover interesting regions in large genomic sequences, by implementing program codes of the algorithm in [5], and testing it using real genomic data to discover all maximal motifs in a sliding window of ℓ-length = 200, on a sequence x of length n.

The application requires the following input parameters. The values for all the parameters are provided by the end user:

- The length n of the string X.
- The length ℓ of the window on X.
- The maximum number d of allowed don't care symbols.
- The minimum number k of occurrences of maximal motifs (Fig. 3).

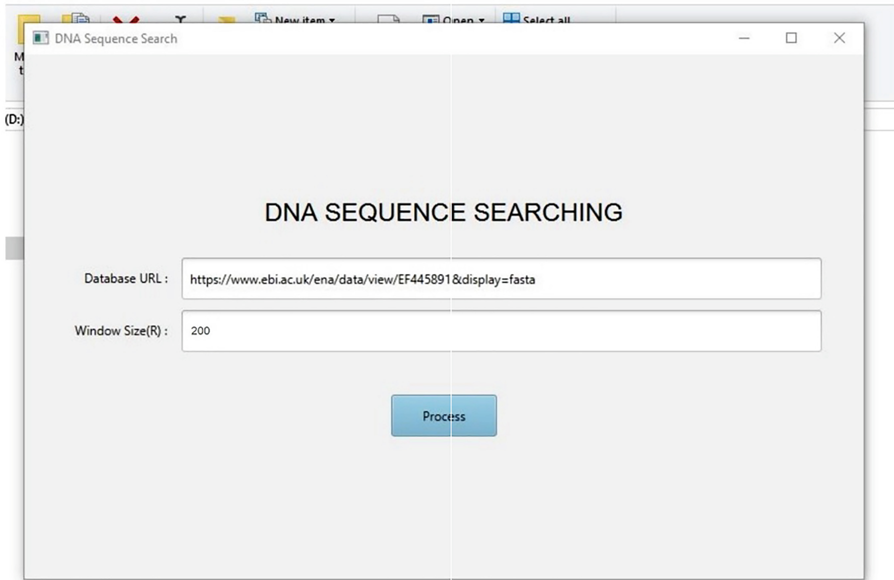
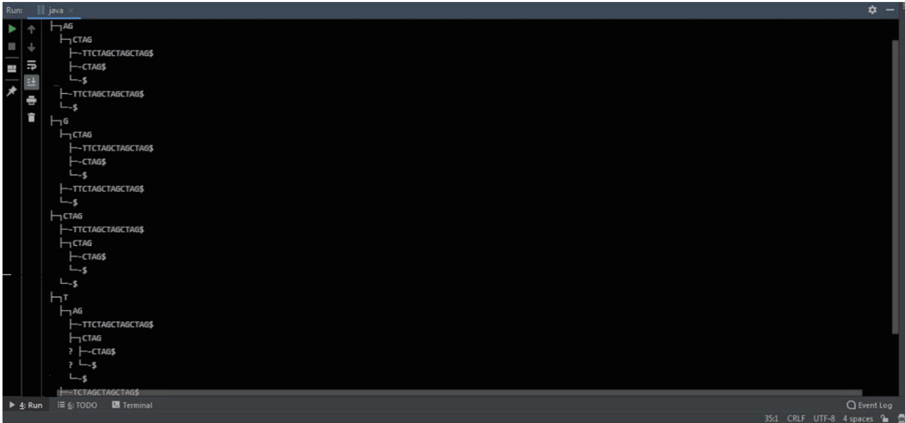


Fig. 3. The application Layout Panel, For the input dataset of the string x, real sequences from the European Nucleotide Archive (ENA) were used. The ENA maintains an extensive record of the world's nucleotide sequencing information, covering raw sequencing data, sequence assembly information and functional annotation. In addition, a database of nucleotide sequences EF445891.1, Escherichia coli strain ATCC 11775 tna operon, was also used [https://www.ebi.ac.uk/ena/data/view/EF445891]. It is important to note that the application utilised could accept any URL of any real genomic dataset input for testing, in order to discover all maximal motifs in a sliding window of ℓ-length = 200.

3.2 Implementation

The outline of maximal motif discovery for biological data in a sliding window can be found in the Appendix.

3.3 Results



In addition to the research outlined, bioinformatic applications similar to the application of the algorithms presented in this paper are outlined below:

- An experimental study of all UniProt protein sequences to discover novel pairs of circularly permuted proteins using algorithm saCSCr.
- An experimental study of bacterial genomes from GenBank to predict the locus of OriC using algorithm MMDSW.
- Algorithm EDSM can be utilised to align reads from newly-sequenced genomesto related pan-genomes.

4 Conclusions and Future Work

In summary, this paper has implemented a bioinformatic application of the motif discovery algorithm, with the purpose of finding biologically significant regions in genomic sequences. In addition, this paper has verified previous theoretical findings by implementing the algorithm in and testing it with real genomic data, in order to discover all maximal motifs in a sliding window of ℓ-length = 200 on a sequence x of length.

With regard to future work, one interesting avenue could be finding maximal motifs that occur a minimum number of times in each sequence, given a set of multiple sequences, somewhat resembling the (ℓ, d)-motif search problem. Furthermore, this paper's implementation of maximal motifs could be extended to gapped maximal motifs. Refinement of these results could take place by restricting the minimum length of a maximal motif or adding a probabilistic post-processing step.

Appendix

```
package Business_Objects;
    import java.io.PrintStream;
    import java.util.ArrayList;
    import java.util.List;
    import java.lang.*;
    public class Suffix {
        private List<Node> nodes = new ArrayList();
        public Suffix(String str) {
           this.nodes.add(new Node());
           for(int i = 0; i < str.length(); ++i) {
              this.addSuffix(str.substring(i));
            }
        }
       public static String fromUnicode(String unicode) {
          String str = unicode.replace("\\", "");
          String[] arr = str.split("u");
          StringBuffer text = new StringBuffer();
          for (int i = 1; i < arr.length; i++) {
            int hexVal = Integer.parseInt(arr[i], 16);
            text.append(Character.toChars(hexVal));
           }
           return text.toString();
        }
       public static String toUnicode(String text) {
          StringBuffer sb = new StringBuffer();
          for (int i = 0; i < text.length(); i++) {
            int codePoint = text.codePointAt(i);
            if (codePoint > 0xffff) {
              i++;
            }
            String hex = Integer.toHexString(codePoint);
            sb.append("\\u");
            for (int j = 0; j < 4 - hex.length(); j++) {
             sb.append("0");
          }
          sb.append(hex);
        }
        return sb.toString();
     }
     private void addSuffix(String suf) {
          int n = 0;
          int n2;
```

```
label37:
for(int i = 0; i < suf.length(); n = n2) {
  char b = suf.charAt(i);
  List<Integer> children = ((Node)this.nodes.get(n)).ch;
  for(int x2 = 0; x2 != children.size(); ++x2) {
    n2 = (Integer)children.get(x2);
    if (((Node)this.nodes.get(n2)).sub.charAt(0) == b) {
String sub2 = ((Node)this.nodes.get(n2)).sub;
      int j;
        for(j = 0; j < sub2.length(); ++j) {
      if (suf.charAt(i + j) != sub2.charAt(j)) {
              int n3 = n2;
              n2 = this.nodes.size();
              Node temp = new Node();
              temp.sub = sub2.substring(0, j);
              temp.ch.add(n3);
              this.nodes.add(temp);
              ((Node)this.nodes.get(n3)).sub  =  sub2.substring
(j);
              ((Node)this.nodes.get(n)).ch.set(x2, n2);
              break;
            }
          }
          i += j;
          continue label37;
        }
      }
      n2 = this.nodes.size();
      Node temp = new Node();
      temp.sub = suf.substring(i);
      this.nodes.add(temp);
      children.add(n2);
      return;
    }
  }
  public void visualize() {
    if (this.nodes.isEmpty()) {
      System.out.println("<empty>");
    } else {
     this.visualize_f(0, "");
    }
  }
  private void visualize_f(int n, String pre) {
List<Integer> children = ((Node)this.nodes.get(n)).ch;
PrintStream var10000;
        Object var10001;
```

```
            if (children.isEmpty()) {
            var10000 = System.out;
            var10001 = this.nodes.get(n);
            System.out.println("\u002d" + ((Node)var10001).sub);
          } else {
            var10000 = System.out;
            var10001 = this.nodes.get(n);
            System.out.println(  fromUnicode("\\u2510")  +  ((Node)-
var10001).sub);
        for(int i = 0; i < children.size() - 1; ++i) {
              Integer c = (Integer)children.get(i);
              System.out.print(pre + fromUnicode("\\u251c\\u2500"));
              this.visualize_f(c, pre + " ");
            }
            System.out.print(pre + fromUnicode("\\u2514\\u2500"));
            this.visualize_f((Integer)children.get(children.size()  -
1), pre + " ");
            }
          }
        }
```

References

1. Carvalho, A.M., Freitas, A.T., Oliveira, A.L., Sagot, M.: Arm efficient algorithm for the identification of structured motifs in DNA promoter sequences. IEEE/ACM Trans. Comput.-Biol. Bioinform. **3**(2), 126–140 (2006)
2. Fuller, R.S., Funnell, B.E., Kornberg, A.: The dnaA protein complex with the E. coli chrornosomal replication origin (oriC) and other DNA sites. Cell **38**(3), 889–900 (1984)
3. Grossi, R., Mermcorri, G., Pisanti, N., Trani, R., Vinc, S.: Motif trie: efficient text index for pattern discovery with don't cares. Theor. Comput. Sci. **710**, 74–87 (2018)
4. van Helden, J., Andre, B., Collado-Vides, J.: Extracting regulatory sites from the upstream region of yeast genes by computational analysis of oligonucleotide frequencies. J. Mol. Biol. **281**(5), 827–842 (1998)
5. Iliopoulos, C.S., Mohamed, M., Pissis, Solon P., Vayani, F.: Maximal motif discovery in a sliding window. In: Gagie, T., Moffat, A., Navarro, G., Cuadros-Vargas, E. (eds.) SPIRE 2018. LNCS, vol. 11147, pp. 191–205. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00479-8_16
6. Leonard, A.-C., Méchali, M.: DNA replication origins. Cold Spring Harb. Perspect. Biol. **5** (10), a010116 (2013)
7. Meijer, M., et al.: Nucleotide sequence of the origin of replication of the Escherichia coli K-12 chromosome. Proc. Nat. Acad. Sci. **76**(2), 580–584 (1979)
8. Pavesi, G., Mercghetti, P., Mauri, G., Pesole, G.: Weeder Web: discovery of transcription factor binding sites in a set of sequences from co-regulated genes. Nucleic Acids Res. **32** (Web-Server-Issue), 199–203 (2004)
9. Pisanti, N., Carvalho, A.M., Marsan, L., Sagot, M.-F.: RISOTTO: fast extraction of motifs with mismatches. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 757–768. Springer, Heidelberg (2006). https://doi.org/10.1007/11682462_69

10. Pissis, S.P.: MoTeX-ll; structured MoTif eXtraction from large-scale datasets. BMC Bioinf. **15**, 235 (2014)
11. Pissis, S.P., Stamatakis, A., Pavlidis, P.: MoTeX: a word-based HPC tool for MoTif eXtraction. In: Gao, J. (ed.) ACM Conference on Bioinformatics, Computational Biology and Biomedical Informatics. ACM-BCB 2013, Washington, 22–25 September 2013, p. 13. ACM (2013)
12. Sinha, S., Tompa, M.: YMF: a program for discovery of novel transcription factor binding sites by statistical overrepresentation. Nucleic Acids Res. **31**(13), 3586–3588 (2003)
13. Waterman, M.S.: General methods of sequence comparison. Bull. Math. Biol. **46**(4), 473–500 (1984)