

Generalized Methods for Discovering Frequent Poly-Regions in DNA

Panagiotis Papapetrou, *Boston University*, Gary Benson, *Member, IEEE*,
and George Kollios, *Member, IEEE*

Abstract

The problem of discovering frequent poly-regions (i.e. regions of high occurrence of a set of items or patterns of a given alphabet) in a sequence is studied, and three efficient approaches are proposed to solve it. The first one is entropy-based and applies a recursive segmentation technique that produces a set of candidate segments which may potentially lead to a poly-region. The key idea of the second approach is the use of a set of sliding windows over the sequence. Each sliding window covers a sequence segment and keeps a set of statistics that mainly include the number of occurrences of each item or pattern in that segment. Combining these statistics efficiently yields the complete set of poly-regions in the given sequence. The third approach applies a technique based on the majority vote, achieving linear running time with a minimal number of false negatives. After identifying the poly-regions, the sequence is converted to a sequence of labeled intervals (each one corresponding to a poly-region). An efficient algorithm for mining frequent arrangements of intervals is applied to the converted sequence to discover frequently occurring arrangements of poly-regions in different parts of DNA, including coding regions. The proposed algorithms are tested on various DNA sequences producing results of significant biological meaning.

Index Terms

Poly-regions, Bioinformatics databases, Data Mining Methods, Mining frequent poly-regions.

I. INTRODUCTION

In cells, DNA forms long chains made up of four chemical units known as nucleotides: adenine (A), guanine (G), cytosine (C), and thymine (T). In these DNA chains or sequences, a number of important, known functional regions, at both large and small scales, contain a high occurrence of one or more nucleotides. We will refer to these as *poly-regions* (for example, a region that is rich in nucleotide A, is called poly-A). Such regions include:

- Isochores. These multi-megabase regions of genomic sequence are specially GC-rich or GC-poor. GC-rich isochores exhibit greater gene density. Human ALU and L1 retrotransposons appear preferentially in isochores with composition that approaches their own [7, 8, 25].
- CpG islands. These regions of several hundred nucleotides are rich in the dinucleotide CpG which is generally underrepresented (relative to overall GC content) in eukaryotic genomes. The level of methylation of the cystine (C) in these dinucleotide clusters has been associated with gene expression in nearby genes [12, 11, 13].
- Protein binding regions. Within these domains, tens of nucleotides long, dinucleotide, or base-step composition, can contribute to DNA flexibility, allowing the helix to change physical conformation, a common property of protein-DNA interactions [24, 19, 14, 18].

Despite the importance of poly-regions, their algorithmic identification and study has received only limited attention.

There has been a variety of approaches and algorithms that consider DNA segmentation. One family of segmentation algorithms employ statistical methods based on: (1) the Maximum Likelihood Estimation (MLE) of the segments. In particular, the MLE is computed for the segments, given a restriction on their minimum length [?]. For the same problem, a dynamic programming approach has been introduced in [?]

that computes the global maximum, whereas [?] proposed an extension where there is no restriction on the segment size, (2) the hidden Markov chain model. Specifically, [?], [?] proposed this idea to model the segmentation of DNA sequences and predict the locations of possible segments in mitochondrial and phage genomes. The model assumes that different segments can be classified into a finite number of states, for example *poly-A*, or *A + T*-rich, (3) the walking Markov model, which is a continuously varying stochastic process. [?] examined the base composition of human and *E.coli* genomes and analyze the phenomenon of strand symmetry, i.e. each base has the same number of occurrences on each strand). They notice the poor fit of Markov models and observe that there is less local homogeneity than necessary for most existing segmentation models. Also, in [?], sequence segments are described using variable length Markov chains, known as tree models (VLMCs). For each segment, a VLMC is obtained that contains the probability distribution vectors that capture the essential features of the corresponding segment, and are finally used to identify segments that closely correspond to the annotated regions of the genes, (4) a Bayesian approach to DNA segmentation is presented in [?], where the Bayesian estimator is used as a measure of homogeneity. The segmentation, carried out via the dynamic programming technique, results in an exact optimal segmentation of the DNA in homogeneous regions.

Simultaneously, there have been studies on similar problems, called “change-point problems” that have been applied to DNA sequence segmentation [?], [?], [?]. The basic form of the multiple change point problem assumes that there exists a set of points in a sequence where the distribution of the sequences changes. Thus, each grouping of consecutive literals (that will form a segment) will arise from a different distribution. The methodology they follow can be broken down into first determining how many change-points exist in a sequence and then finding their locations. Also, in [?], a study on change-points (transitions between homogeneous and inhomogeneous regions of DNA) is carried out, and rigorous methods of information theory are employed to quantify structural properties of DNA sequences.

Another family of DNA segmentation algorithms includes those that work in a hierarchical manner (top-to-bottom). In particular, they employ recursive segmentation of DNA sequences, where at each stage a split point is chosen based on a specific criterion, e.g. the Jensen-Shannon Divergence [?], [?]. Such algorithms have been proposed in [?], [?], [?], [?] and their main focus was to find domains in DNA that are homogeneous in base composition or more specifically in C+G content. Moreover, in [?], [?] it is shown that there are many other applications of the recursive segmentation algorithm to the analysis of DNA sequences, such as detection of isochores (large homogeneous C+G domains), CpG islands (small homogeneous CG domains), etc. Another recursive segmentation approach is presented in [?], where the DNA sequence is divided into compositionally homogeneous domains by iterating a local optimization procedure at a given statistical significance. Once the DNA sequence is partitioned into domains, a global measure of sequence compositional complexity (SCC), accounting for both the sizes and compositional biases of all the domains in the sequence, is derived. The algorithm computes SCC as a function of the significance level, which provides a multiscale view of sequence complexity.

Last but not least, a sliding window approach with fixed size window has been applied on the human genome [?], [?] to detect *G + C*-rich regions and CpG islands. Also, in [?], a reliable segmentation method is used to partition the longest contigs in the human genome into long homogeneous regions (LHGRs), thereby revealing the isochores.

To the best of our knowledge, all current approaches target specific compositions (mainly *G + C*-rich or CpG islands) and also they do not examine specific genome areas, e.g. introns, exons, etc. At the same time, there has been a lot of work on nucleosome positioning in DNA sequences [?]. In particular, it has been observed that nucleosome regions contain some DNA signals (patterns) that can be used to identify these types of regions. However, there have been no studies on any specific types of poly-regions that might occur between different poly-regions that could be related to nucleosomes. Being able to identify some standard patterns or types of poly-regions that occur over a nucleosome may be further used to infer nucleosome positions. Furthermore, there have been no studies on relations that may occur between these regions. The first set of algorithms for detecting poly-regions of more general composition has been proposed in [?]. In this paper, we provide a more general and robust definition of poly-regions and

describe three efficient approaches to finding any type of poly-region in DNA. We further apply efficient mining techniques to extract frequent patterns in those regions.

The main contributions in this paper include:

- a formal definition of the problem of discovering poly-regions of items or patterns in a sequence.
- an exact algorithm that uses a set of sliding windows over the sequence.
- two approximate algorithms for detecting poly-regions: the first one is entropy-based and uses recursive segmentation techniques and the second one is based on the majority vote.
- the application of an efficient arrangement mining algorithm to extract the complete set of frequent arrangements of these poly-regions.
- an extensive experimental evaluation of our algorithms by testing their efficiency on the Dog genome.
- an analysis of some standard types of poly-regions that have been detected on exons, introns and nucleosomes in various DNA sequences of the Dog and Yeast genomes.

II. PROBLEM FORMULATION

A sequence $\mathcal{T} = t_1 t_2 \dots t_m$ is an ordered set of items, where each t_i belongs to an alphabet Σ . In the case of DNA sequence, each t_i corresponds to a nucleotide base and thus $\Sigma = \{A, C, G, T\}$, where A stands for *Adenine*, C for *Cytosine*, G for *Guanine* and T for *Thymine*. A *poly-region* $P = \{\mathcal{X}, start, end\}$ is a segment of \mathcal{T} , where there is a “high occurrence” (which is defined below) of \mathcal{X} , that starts at item t_{start} and ends at item t_{end} . \mathcal{X} determines the type of the poly-region, and a poly-region with $|\mathcal{X}| = k$ is called *k-poly-region*. In this paper, two types of poly-regions are considered:

- 1) **Poly-region of Type I:** $P = \{\mathcal{I}, start, end\}$, where $\mathcal{I} \subseteq \Sigma$ is a set items, with $t_{start} \in \mathcal{I}$ and $t_{end} \in \mathcal{I}$. Examples of poly-regions of Type I are: poly- $\{A\}$ (known as poly-As), poly- $\{A,C\}$ (known as poly- $\{A+C\}$ s), etc.
- 2) **Poly-region of Type II:** $P = \{\mathcal{S}, start, end\}$, where $\mathcal{S} = s_1 s_2 \dots s_{|\mathcal{S}|}$ is a sequence of items, with $s_i \in \Sigma$, $t_{start} = s_1$ and $t_{end} = s_{|\mathcal{S}|}$. An example of a poly-region of Type II is: poly- $\{CG\}$ (also known as *CpG-island*).

Given a Type I poly-region $P = \{\mathcal{I}, start, end\}$, with $|\mathcal{I}| = k$, the frequency of each item i in P is defined as

$$f_i = \frac{\# \text{ of occurrences of } i \text{ in } P}{|P|}.$$

P has density d , if $\sum_{i=1}^{|\mathcal{I}|} f_i \geq d$ and for each $i \in \mathcal{I}$, $f_i \geq \frac{d}{2k}$. This means that the sum of the individual frequencies of each item should be at least d and each individual frequency should be at least $\frac{d}{2k}$. For example a poly- $\{A+C\}$ of size 20 should have at least $20 \frac{d}{4}$ As, at least $20 \frac{d}{4}$ Cs and the sum of As and Cs should be at least $20d$.

In the case of a Type II poly-region $P = \{\mathcal{S}, start, end\}$, the frequency of \mathcal{S} is defined as

$$f_{\mathcal{S}} = \frac{\# \text{ of occurrences of } \mathcal{S} * |\mathcal{S}|}{|P|}.$$

\mathcal{P} has density d , if $f_{\mathcal{S}} \geq d$.

Given a density threshold *min_density*, a poly-region of density d is said to be *dense*, if $d \geq \text{min_density}$. In Figure ??, we can see four examples of poly-regions: (1) is a poly-A region, with $P = \{\{A\}, 5, 14\}$ with density 80%, (2) is a poly- $\{A,C\}$ region, with $Q = \{\{A, C\}, 20, 29\}$, where each one has a frequency of 40%, (3) is a poly-ApC, with $P = \{\{AC\}, 32, 39\}$ and density 75%, and (4) is a poly-CpT, with $P = \{\{CT\}, 49, 60\}$ and density 91%.

Given two poly-regions P and Q , $P = \{\mathcal{X}, p_{start}, p_{end}\}$, and $Q = \{\mathcal{Y}, q_{start}, q_{end}\}$, the *merging* of P and Q is a new poly-region P' , with

$$P' = \{\mathcal{X}, \min\{p_{start}, q_{start}\}, \max\{p_{end}, q_{end}\}\}.$$

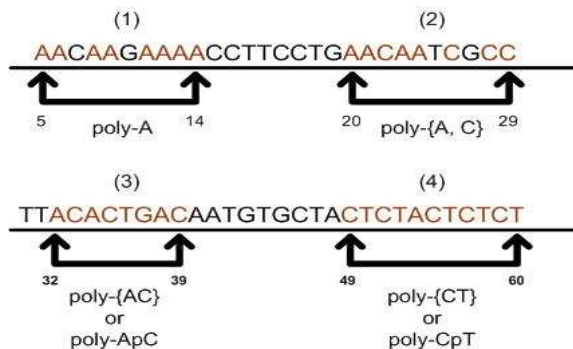


Fig. 1. Example of two Poly-regions.

Notice that merging is only allowed when $\mathcal{X} = \mathcal{Y}$. Also, a poly-region $P = \{\mathcal{X}, p_{start}, p_{end}\}$ is said to be contained in another poly-region $Q = \{\mathcal{Y}, q_{start}, q_{end}\}$, if $q_{start} \leq p_{start}$, $q_{end} \geq p_{end}$, and $\mathcal{X} = \mathcal{Y}$. A dense poly-region P with density d_1 is *maximal*, if there exists no poly-region Q with density d_2 such that $d_2 \geq min_density$ and P is contained in Q .

A poly-region can be seen as an *event interval*, which (based on [?]) is a triple $(e_i, t_{start}^i, t_{end}^i)$, where e_i is an event label, t_{start}^i is the start position of the event and t_{end}^i is the end position in the DNA sequence. A set of event intervals, ordered by their start time, is called an *event interval sequence* or *e-sequence*. Thus, a set of poly-regions of a DNA sequence \mathcal{T} constitutes the corresponding e-sequence of \mathcal{T} . A more detailed analysis on the above terminology and concepts is given in section ??.

Our goal is to first find the complete set of poly-regions given an input sequence and then apply an efficient algorithm for mining frequent arrangements of temporal intervals [?] to discover arrangements of poly-regions that occur frequently (1) in the sequence, (2) among different segments of the sequence. For (2), the Hybrid-DFS algorithm described in [?] applies directly, whereas for (1) an approach similar to that described in [?] for mining frequent episodes over a sequence of instantaneous events can be employed.

Problem Statement: Given a sequence $\mathcal{T} = t_1 t_2 \dots t_m$, a density constraint d , a minimum window size min_win , a maximum window size max_win and a support threshold min_sup , are goal is to:

- 1) discover the complete set \mathcal{P}_S of maximal poly-regions in \mathcal{T} , where each region has density of at least d and size $\in [min_win, max_win]$, and then
- 2) given \mathcal{P}_S , define a set of segments of \mathcal{T} (in the DNA case, these segments could for example be coding regions), and based on a support threshold min_sup , extract the complete set \mathcal{F} of arrangements of poly-regions that occur frequently in those segments.

III. EXTRACTING POLY-REGIONS

In this section we present three approaches for extracting the set of poly-regions in a sequence of items that belong to a given alphabet. For our purposes, since we are dealing with DNA sequences where the alphabet is of size four, we only focus on the following poly-regions, given $\Sigma = \{A, C, G, T\}$:

- 1) all poly-regions $P = \{\mathcal{I}, p_{start}, p_{end}\}$ of Type I, with:
 - a) $|\mathcal{I}| = 1$ (which gives a total of K_1 poly-regions, with $K_1 = |\Sigma|$).
 - b) $2 \leq |\mathcal{I}| \leq 3$ and all items in \mathcal{I} are different from each other (which gives a total of K_2 poly-regions, with $K_2 = \frac{(|\Sigma|+1)|\Sigma|}{2}$).
- 2) poly-regions $P = \{\mathcal{S}, p_{start}, p_{end}\}$ of Type II, where $2 \leq |\mathcal{S}| \leq 3$. Notice that in the case where $|\mathcal{S}| = 2$, the two nucleotides should be different from each other, and in the case where $|\mathcal{S}| = 3$, the case where all three nucleotides are the same is excluded (which gives K_3 poly-regions, with $K_3 = |\Sigma| (|\Sigma| - 1) + |\Sigma| (|\Sigma|^2 - 1)$).

The first approach is entropy-based and uses an existing recursive segmentation technique to split the input sequence into a set of homogeneous segments applying measures of divergence (in our case the *Jensen Shannon Entropy*) during the segmentation, the second one implements a set of sliding windows over the sequence, and the third uses a technique based on the majority vote. Notice that the second approach is exact and produces the complete set of poly-regions, whereas the other two methods are approximate.

A. Recursive Segmentation

The idea of recursive segmentation based on a measure of divergence has been used in earlier works [?], [?], [?], and [?] describes how it can be applied to DNA for detection of $G+C$ -rich regions and CpG islands. In this section we present an approach that applies the standard recursive segmentation algorithm used previously targeting poly-regions of Type I and II. The main difference in our approach is that the recursive segmentation does not use the standard stopping criterion [?]; instead, the recursion stops when the size of a segment drops below *max_win*.

More specifically, the input sequence is recursively segmented, ensuring that the homogeneity difference (in our case the entropy) between the segments is maximized. To define the homogeneity difference between two segments, an appropriate measure λ is used. There is a variety of measures that can be used for the segmentation process, like the quadratic divergence (QD) [?], the Jensen-Shannon Divergence (JSD) [?], the Gini-Index, etc. In this work, we use the *Jensen-Shannon divergence*.

The target of the segmentation is a set of regions, where, in each region, the *Jensen-Shannon Entropy* is maximized. To achieve that, the input sequence is recursively segmented and each time a split point is chosen where the JSD value between the two segments is maximized, i.e. the distributions of the items in the two segments have maximal JSD value from each other. The recursive segmentation stops for a segments, when it is of size from *min_win* to *max_win*. The final segmentation includes a set of regions of the desired size that are candidates for being poly-regions. Through a sequential scan of each segment these regions are identified by checking whether there exists a set of elements (of poly-regions of Type I or Type II) that satisfies the density constraint in that segment.

Before proceeding to a detailed description of the algorithm, let us first give some basic definitions. Let $S = \{s_1, s_2, \dots, s_m\}$ be the input sequence and $P(S) = [l..r]$ be a segment of S , i.e. the subsequence of S starting at s_l and ending at s_r , for $1 \leq l, r \leq m$. A segmentation of S is denoted as $S_g = \{n_1, n_2, \dots, n_{M-1}\}$, where each n_i is an index of a point in S . Trivially, S_g defines M segments, where each segment starts at point $s_{n_{j-1}}$ and ends at point s_{n_j} , with the first segment starting at point s_1 and ending at point s_{n_1} and the last segment starting at point $s_{n_{M-1}}$ and ending at point s_m . Given a segmentation $P(S)$ of S , $f(P) = \{f_i, i = 1, \dots, t\}$ denotes the set of frequencies of each item in $P(S)$, where

$$f_i = \frac{\text{number of occurrences of item } i \text{ in } S}{|S|}$$

and t is the number of distinct items.

Let $H = -\sum f_i \log_2 f_i$, for $i = 1, \dots, t$ be the Jensen-Shannon Entropy of a sequence S , where f_i is the frequency of item i in S . Then, the Jensen-Shannon Divergence of two segments $P = [1..n]$, $Q = [(n+1)..m]$ of S is defined as

$$D(n) = H - \left(\frac{n}{m} H_{left} + \frac{m-n}{m} H_{right} \right)$$

where H_{left} and H_{right} denote the Jensen-Shannon entropy for the left and right subsequences respectively.

Next, the algorithm is presented in more detail. The main characteristic of the algorithm is that it recursively splits the input sequence until the final segmentation is reached, where each segment is of maximal homogeneity. Finally, the segments are scanned to extract the set of poly-regions.

1) *The Algorithm in Detail:* Starting with the original sequence S , the algorithm looks for the index $n \in [1, |S|]$ of S that maximizes the JSD value of the two segments $P = S[1..n]$ and $Q = S[(n+1)..m]$. The same process is applied recursively to each segment until a halting condition is satisfied. In our case, the halting condition requires that each segment should be of length between min_win and max_win . Thus, given a segment P , if the next step of the segmentation produces segment of length less than max_win , the recursion stops and P is reported, since it is a candidate poly-region; otherwise the recursive segmentation is continued. In the case where the new segment is of size less than min_win , the recursion again stops but without reporting the segment.

The algorithm as described above can efficiently detect regions of high occurrence of a nucleotide base. However, if we are interested in poly-regions of more than one nucleotides or poly-regions of Type II, the above process may fail. To achieve an efficient segmentation for both types of poly-regions, a preprocessing step is applied, which has been suggested in [?] for the detection of isochores. When looking for poly-regions of two nucleotides, say poly- $\{W, Y\}$, the original sequence is transformed to a new sequence as follows: each W and Y nucleotide is replaced by literal X , whereas the rest are replaced by a literal taken from Σ (each time a different literal is chosen and when all literals of Σ have been used, we start over). For example, if $S = ACAAAGCGA$ and we are looking for poly-regions of A , S will be converted to $S' = XAXXXCGTX$, given that $\Sigma = \{A, C, G, T\}$. The same idea is followed when looking for all poly-regions of Type I. As for poly-regions of Type II, the input pattern is detected in the sequence and all the literals that are part of the pattern are changed to X , whereas the other ones are replaced as in the case of Type I poly-regions. The benefit of this replacement is the following: at each step of the segmentation, two regions are under consideration, say r_1 and r_2 . If r_1 is of high occurrence of the desired pattern and in r_2 (which is the rest of the sub-sequence under consideration) all literals are different, the entropy difference between r_1 and r_2 will be maximized.

The steps of the recursive segmentation algorithm are given below:

- 1) Given an input sequence S , for each type of poly-region, S is converted to S' as described above.
- 2) Given S' , $JSD(P, Q)$ is calculated, with $P = S[0..n]$ and $Q = S[n+1..m]$, for each $n \in [2, m-1]$.
- 3) Let n be the index of S' where λ (in our case JSD) is maximized. S' is segmented, and the index n is reported. If the halting condition is satisfied for a segment, the segmentation process terminates for that segment, otherwise it proceeds recursively.
- 4) When the above process is completed, a segmentation $S_g = \{n_1, n_2, \dots, n_{M-1}\}$ of M segments is generated. Each of these segments is a candidate poly-region. Next, a linear scan is performed on S_g . Each segment is checked whether it satisfies the density constraint and it is further expanded both ways until the density constraint is violated. When a poly-region is found it is reported.

2) *Complexity:* Every time the sequence is split into two subsequences. The number of splits is $O(\log(|S|/(max_win - min_win)))$, where $|S|$ is the size of the original sequence. Since on each recursion each segment is read once and at the final step we just perform a linear scan, the total runtime of each run of the algorithm is $O(|S|\log|S|)$. Now, given that the alphabet size is Σ , the number of times the algorithm is run is K' , the total runtime of the algorithm is $O(K'|S|\log|S|)$, and since K' is a constant (and $K' \ll |S|$), this becomes $O(|S|\log|S|)$.

B. Sliding Windows

The key idea behind this approach is to use a set of sliding windows over the input sequence. Each sliding window keeps statistics of a segment that mainly include the number of occurrences of each candidate element (meaning each item or sequence of the poly-regions we are looking for) in that segment. Combining these statistics efficiently produces the complete set of poly-regions in the sequence.

More formally, our algorithm is given a sequence S , a density factor d , a minimum window size min_win and a maximum window size max_win . The first step is to define a set of sliding windows \mathcal{W} .

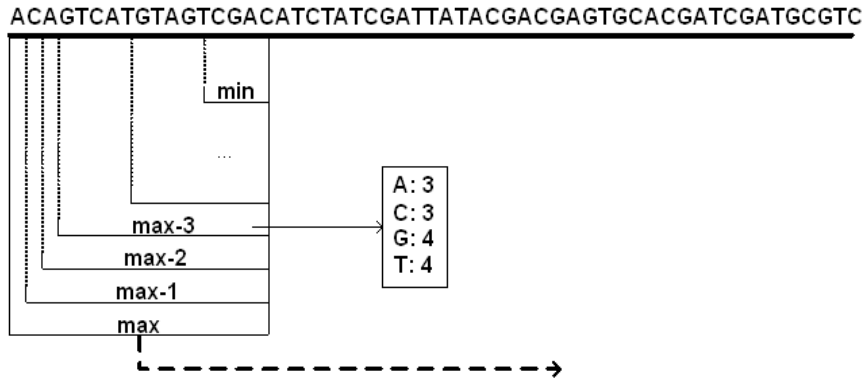


Fig. 2. An Example of an Instance of the Set of Sliding Windows

Let $\mathcal{W} = \{w^1, w^2, \dots, w^n\}$, where w^i corresponds to sliding window i and $n = |\mathcal{W}| = max_win - min_win + 1$. Each sliding window w^i is a triplet $\{C^i, w^i_{start}, w^i_{end}\}$, where C^i is a set of statistics for w^i , w^i_{start} is an index to the starting position of w^i on S and w^i_{end} is an index to the ending position of w^i on S . C^i is a set of t counters $\{C_1, C_2, \dots, C_t\}$, with $t = K_1 + K_3$ or $t = K'_1 + K'_3$ if reverse complements are excluded. The value of each counter is the number of occurrences of the corresponding item/sequence in the window. Moreover, the piece of S covered by \mathcal{W} is stored at each time instance. An example of an instance of the set of sliding windows used by the algorithm is shown in Figure ?? . Given this setting, at any time, we can extract the top k frequent items in each window.

The next step is to identify the set of poly-regions using \mathcal{W} . More specifically, all the windows defined in \mathcal{W} will be sliding simultaneously. Conceptually, the above setting can be seen as having $M = max_win - min_win + 1$ levels of windows, one for each size. At any time instance, we check the statistics stored under each window in \mathcal{W} . If a set of items or sequences in a window w^i is found to satisfy the density constraint, then w^i is reported as a poly-region. In parallel, we keep a list L of the poly-regions discovered so far. Each record in L corresponds to a poly-region label and points to a list of all the poly-regions discovered so far with this label. Upon discovery of a new poly-region we insert it into L based on its label.

Notice that the sliding window approach makes sense when the alphabet size is small, which holds for the application this paper is focused on, i.e. the DNA alphabet size is only four.

1) *The Algorithm in Detail:* The algorithm has three phases: the Initialization Phase, the Sliding Phase and the Merging Phase. During the first phase, \mathcal{W} is initialized; this phase is completed as soon as the first max_win characters of the sequence are read. Then the algorithm proceeds with the Sliding Phase, where \mathcal{W} slides across the sequence until it reaches the end of the sequence. Before inserting each new poly-region into L the Merging Phase is activated, to identify any old poly-region that can be absorbed by the new one. More details on the three Phases are given below:

- 1) Initialization Phase: the first min_win characters are read and window w^1 is created. This is in fact the window of the smallest size in \mathcal{W} . The counters of w^1 are updated based on what has been read so far. For each new character s_j , a window w^i , for $i = 2, \dots, n$, of size $min_win + i - 1$ is created starting at character s_1 and ending at character s_j . The counters of each window w^i are updated based on the counters of the previous window (i.e. w_{i-1} . Let C_j^{i-1} , for $j = 1, \dots, t$ denote the counters of the $(i-1)$ -th window. Then $C_j^i = C_j^{i-1}$, for $j = 1, \dots, t$. This process is repeated until $j = max_win$. Every time a new window is created and all the counters are updated, the window is checked for items that satisfy the density constraint. If so, it constitutes a poly-region and is added into L after applying the Merging Phase. Upon completion of the current phase, \mathcal{W} has been fully created. Notice that in this phase, no sliding is performed on the windows.
- 2) Sliding Phase: during this phase, \mathcal{W} keeps sliding to the right and for every new item s_i , the

corresponding counters are updated, i.e. for each w^i in \mathcal{W} , $C_{s_i} = C_{s_i} + 1$. Since each window in \mathcal{W} is moved one position to the right, the counter of the element that is no longer in the window has to be decreased by one, i.e. for each w^i in \mathcal{W} , $C_{S_{start}} = C_{S_{start}} - 1$. Finally, the start and end pointers of each window are updated accordingly. After a slide is performed and all counters are updated, each window is checked for having any itemset or sequence satisfying the density threshold. Starting with the window of maximum size, if element c is found to satisfy the density threshold, then this window is reported as a poly-region of c . Since we are only looking for maximal windows, the counter of c is not checked any more in the rest of the windows in the current instance of \mathcal{W} . Finally, each poly-region is added into L after applying the Merging Phase.

- 3) Merging Phase: for each new window w^j , before it is inserted into L , the corresponding record of L is scanned for a window w^i such that the start points of w^i and w^j coincide and w^i is contained in w^j . Trivially, if such window exists, it will be one of the last $max_win - min_win + 1$ inserted in that record. Before the insertion of w^j in L , w^i is removed. Also, since the windows inserted into L are ordered by their start time, if a window is reached, with start point smaller than that of w^j , then the process stops and inserts w^j in L .

Notice that at each step we do not need to check all the windows. Instead we can start with the window of maximal size and prune some of the smaller windows. More specifically, the value of each counter in a large window is an upper bound for the value of the corresponding counters in the smaller windows in \mathcal{W} . Let the number of elements of type c (either itemsets or sequences) in w^i be N_c^i . Then c is dense in w^i , if $\frac{N_c^i}{|w^i|} \geq d$. Hence, the maximum size of the window were these elements (of type c) can fit and fulfill the density constraint is $\frac{N_c^i}{d}$. Based on this observation, we can start with the maximum window and then apply the bound on each counter. This indicates which windows of the lower levels should be searched for a candidate poly-region for each item. Consider Figure ??(2) for example, and let $d = 50\%$. Suppose that $max_win = 10$, and currently the maximum window in \mathcal{W} is the DNA sequence segment shown in the Figure and notice that $C_c = 4$. Then the maximum window in \mathcal{W} , where item C can be dense, is of size $\frac{C_c}{d} = 8$. Thus, in order to look for a poly-region of nucleotide C , we should skip w^9 . The described method produces a set \mathcal{P}_S of poly-regions for the input sequence S .

2) *Complexity*: Based on the previous analysis, it can be seen that at any time instance, the number of windows under consideration is $M = max_win - min_win + 1$. Moreover, for each window we keep t counters, which yields a total of tM counters. Also, for each set of windows \mathcal{W} we store the piece of the sequence that is covered by the maximum window. Thus, the space complexity is $O(|\Sigma|M + max_win)$. Each element is read once and then stored in \mathcal{W} . At each slide, in the worst case M windows are accessed. For each window, the value of t counters is checked and the last element of each window is removed. Therefore, for each slide a total of Mt counters are accessed. Also, when a window is determined to constitute a poly-region, at most M records are accessed in the list L to check whether it overlaps with an existing poly-regions. The above analysis yields a time complexity of $O(|S|M)$. Since in practice $max_win, min_win \ll |S|$, the algorithm is linear.

C. Majority Vote

Another efficient approach is described in this section that employs the idea of the majority vote, first used in [?] for finding repeated items in a sequence. The same concept was later used in [?] for finding frequent items over sliding windows. Our goal is to improve the performance of the sliding window algorithm by having only a single sliding window w along with: (1) a set of *primary counters* C_p and (2) a set of *secondary counters* C_s . The primary counters are used to indicate regions that are candidate poly-regions. If a candidate poly-region is detected, then the set of secondary counters is examined to check if it actually is a poly-region.

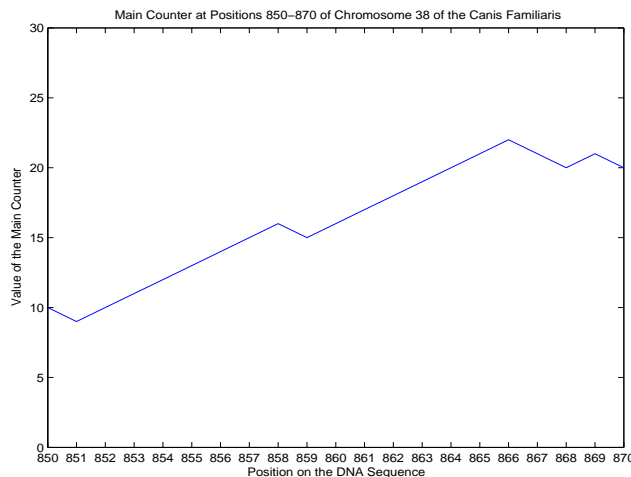


Fig. 3. Value of the Main Counter at positions 850 – 870 of Chromosome 38 of the Canis Familiaris.

In particular, the algorithm uses $t_1 = 3 + K_2$ primary counters and $t_2 = K_1 + K_3$ secondary counters, along with a set of buffers holding the literal corresponding to each primary counter. All counters are initially set to zero, the first literal of the input sequence is read and stored under the right buffer, and the corresponding primary counter is increased by one. Each time a new literal is read, the sequence index is increased by one. If the new literal matches one stored under a buffer, then the corresponding primary counter is increased by one, otherwise it is decreased by one; if a primary counter reaches zero, the literal currently in its buffer is replaced by the new one. In any other case, we move on to the next literal in the sequence. When an element (either itemset or sequence) is identified in the sequence the corresponding secondary counters are updated so that, at any time during the sequence scan, each secondary counter is equal to the number of occurrences of the corresponding element in the window. This process continues until the whole sequence is read. In the case of Type I poly-regions of a set of literals, all literals in the set are considered to be the same during the scan. As for poly-regions of Type II, they can be seen as a single literal. For example, consider the sequence $ACACCAC$. In this case, the first literal is AC , the next is CA , the next ACA , then CA and so on. This explains the need for more than one primary counters: 1 counter for the single items, K_2 counters for the itemsets, 1 counter for poly-regions of Type II with $|\mathcal{X}| = 2$ and one for poly-regions of Type II with $|\mathcal{X}| = 3$, yielding a total of t_1 primary counters.

The benefit of this approach is that the behavior of the primary counters can imply high occurrence of a set of items or subsequence in a specific region of the sequence. In fact, we have two cases: (1) if a primary counter increases rapidly, then there is high occurrence of the corresponding literal stored in the buffer implying the existence of a poly-region, (2) if a primary counter decreases rapidly, then the corresponding literal in the buffer does not occur frequently in that region. Instead another literal might be in majority in the region, which will constitute a poly-region. However, this might not be the case since decrease on a primary counter only implies that the corresponding literal in the buffer is not in majority in that area and does not necessarily imply majority of another literal. In Figure ??, we can see the behavior of a primary counter at a poly-region on chromosome 38 of the *Canis Familiaris*. Notice that in this case a poly-region of nucleotide C has been reported at [852, 866].

Let w be the sliding window, and c_i^S be the value of counter i at the beginning of w , and c_i^E be the value of the counter at the end of w . The following lemmas hold, based on the previous analysis for each primary counter.

Lemma 1: If $\exists C_i \in \mathcal{C}_p$ such that $\Delta C_i > 0$ and $\Delta C_i \geq |w|(2d - 1)$, where d is the density constraint, then w corresponds to a poly-region.

Proof: Since w corresponds to a poly-region \mathcal{P} of say element c , the number of occurrences of c in P is $N_c \geq |w|d$. The counter will be increased by one at least $|w|d$ times, and it will be decreased by one at most $|w|(1-d)$ times. Thus, the total change of the counter in a poly-region can be at least $|w|(2d-1)$.

Lemma 2: If $\exists C_i \in \mathcal{C}_p$ such that $\Delta C_i < 0$ and $|\Delta C_i| \geq |w|(2d-1)$, then w is a candidate poly-region.

Proof: Lemma 2 is proved by an argument similar to that for Lemma 1. However, in this case, since the fact that the counter decreases does not necessarily mean that the same literal appears consecutively. Thus, w corresponds to a candidate poly-region.

Lemma 3: If for all counters $C_i \in \mathcal{C}_p$, $|\Delta C_i| < |w|(2d-1)$, then w cannot be a poly-region.

Proof: Straightforward, from the above Lemmas.

The algorithm applies the above lemmas each time w slides to the right. Every time a poly-region is discovered by Lemma 1, it is added into the set of poly-regions. When a candidate poly-region is discovered by Lemma 2, the set of secondary counters in w are invoked to check whether it actually corresponds to a poly-region and if not it is discarded.

1) *The Algorithm in Detail:* Let w be the sliding window, and c_i^S be the value of counter i at the beginning of w , and c_i^E be the value of the counter at the end of w . Also, let the change on a counter be $\Delta C_i = c_i^E - c_i^S$. The main steps of the algorithm are the following:

- 1) If for all primary counters in \mathcal{C}_p , $\Delta C_i < |w|(2d-1)$, slide to the right.
- 2) If $\exists C_i \in \mathcal{C}_p$ such that $\Delta C_i > 0$ and $\Delta C_i < |w|(2d-1)$, then w is reported as a poly-region.
- 3) If $\exists C_i \in \mathcal{C}_p$ such that $\Delta C_i < 0$ and $|\Delta C_i| \geq |w|(2d-1)$, then w is a candidate poly-region. Each of the secondary counters is checked. If for a set of secondary counters $C' \subseteq \mathcal{C}_s$, $\Delta C^j = C_j^E - C_j^S \geq |w|d$ ($\forall j \in C'$), then w is reported as a poly-region of C' .
- 4) Steps 1-3 are repeated until the whole sequence is scanned.

Finally, we get a set of poly-regions of size $|w|$. However, according to the problem formulation, the poly-regions should be of size min_win to max_win . To capture all these regions, we set $|w| = \frac{min_win}{2}$ and when a poly-region is detected, it is expanded as much as possible in order to detect all the maximal legal poly-regions in the range of $[min_win, max_win]$ in that area, keeping in mind that a valid poly-region should start and end with specific literals (a poly-A should start and end with an A). This step is the most costly one of this method. Notice that once a poly-region of size $|w|$ is discovered, the expansion should make sure that the final poly-region will: (1) include w , (2) be of size $\in [min_win, max_win]$, (3) start and end with the appropriate literals. To satisfy the third condition efficiently, an index of each literal in Σ is built at the beginning of the algorithm, such that for each literal in Σ we get to know the positions where it occurs in the sequence. This requires a single scan, and the indices are stored in $|\Sigma|$ arrays, one for each literal. Also, for each array we keep a pair of pointers that move according to w : while w scans the sequence the pointers slide over the indices so that they include the positions where each literal occurs in that part of the sequence currently under w . To satisfy the second condition we need to expand w both ways. Since $|w| = \frac{min_win}{2}$, we need to check $r = max_win - |w|$ positions to the right and to the left. Since w has to be fully contained in the larger poly-region a maximum of r^2 checks is needed: we have r candidate positions on the left and r positions on the right and we check their combinations. Notice that since the maximum poly-region size is bounded by max_win , we can skip some of the above checks, i.e. we first check the poly-region that starts at point $w_{start} - r$ and ends at w_{end} , then the poly-region starting at point $w_{start} - r + 1$ and ending at w_{end} , and so on. If one of those windows is a valid poly-region we check for any possible merging with any other region found in this step and then report the new poly-region.

2) *Complexity*: In terms of space complexity, the algorithm is efficient, since it only needs to keep two pointers (one to the start and one to the end point of the window w), a total of t_1+t_2 counters, a set of t_1+t_2 buffers, and $|S|$ index values. Regarding time complexity, one sequence scan is needed to make the indices, and for each small poly-region (of size $|w|$) we need to check at most $O(r^2)$ expansions. This gives a total cost of $O(|r^2||S|)$. Notice that $r = \text{max_win} - |w|$ and since in practice $\text{max_win}, \text{min_win} \ll |S|$, the algorithm is linear.

IV. DISCOVERING FREQUENT ARRANGEMENTS OF POLY-REGIONS

In this Section we apply an efficient algorithm [?] for mining frequent arrangements of poly-regions on a single input sequence of event intervals.

A. Background

Let $\mathcal{E} = \{E_1, E_2, \dots, E_m\}$ be an ordered set of event intervals, called *event interval sequence* or *e-sequence*. As seen previously, each E_i is a triple $(e_i, t_{start}^i, t_{end}^i)$, where e_i is an event label, t_{start}^i is the event start point and t_{end}^i is the end point. The events are ordered by the start point. If an occurrence of e_i is instantaneous, then $t_{start}^i = t_{end}^i$. An e-sequence of size k is called a *k-e-sequence*. If the first event interval in an e-sequence of size m starts at point t_{start}^1 and the last event interval in the e-sequence ends at point t_{end}^m , then the *width* of the e-sequence is equal to $t_{end}^m - t_{start}^1 + 1$.

An arrangement \mathcal{A} of n events is defined as $\mathcal{A} = \{\mathcal{E}, R\}$, where \mathcal{E} is the set of event intervals that occur in \mathcal{A} , with $|\mathcal{E}| = n$, and $R = \{R(E_1, E_2), R(E_1, E_3), \dots, R(E_1, E_n), R(E_2, E_3), R(E_2, E_4), \dots, R(E_2, E_n), R(E_{n-1}, E_n)\}$. R is the set of relations between each pair (E_i, E_j) , for $i = 1, \dots, n$ and $j = i + 1, \dots, n$, and $R(E_i, E_j) \in \mathcal{R}$ defines the relation between E_i and E_j . The size of an arrangement $\mathcal{A} = \{\mathcal{E}, R\}$ is equal to $|\mathcal{E}|$. An arrangement of size k is called a *k-arrangement*. Given an e-sequence s , s contains an arrangement $\mathcal{A} = \{\mathcal{E}, R\}$, if all the events in \mathcal{A} also appear in s with the same relations between them, as defined in R .

What remains to be defined are the types of relations that are going to be considered. Seven types of relations between two event intervals are considered. Using these relations, general arrangements can be defined. However, our methods are not limited to these relations and can be easily extended to include more types of relations, such as the the ones described in [?], [?]. In Figure ?? we can see the types of relations considered in this paper. More details are given in [?].

B. Mining Frequent Arrangements in a Single DNA Sequence

In this Section we describe an efficient algorithm for mining frequent arrangements of intervals on a single e-sequence S . The algorithm uses a sliding window w of size win to scan the whole e-sequence. w is initially placed at the beginning of the e-sequence and includes the first win event intervals (in our case poly-regions) of S . The window keeps sliding to the right (one event interval per slide) until it reaches the end of S , i.e. its right end includes the last event interval of S , for the first time. Based on this formulation, a total of $\mathcal{W} = |S| + \text{win} - 1$ windows is defined over the sequence. The frequency of an arrangement \mathcal{A} is defined as the fraction of windows in which \mathcal{A} occurs. Thus, given \mathcal{A} and a window of size win , the frequency of \mathcal{A} is: $\text{freq}(\mathcal{A}, \text{win}) = \frac{|\{w|\mathcal{A} \text{ occurs in } w\}|}{|\mathcal{W}|}$.

1) *The Algorithm in Detail*: The algorithm uses the *arrangement enumeration tree* structure, introduced in [?], which is traversed in a DFS manner. The discovered arrangements are stored in a list L , along with their frequencies. In each window w , the set of arrangements contained in w is identified, and the list of active arrangements L is updated. If a new arrangement is found, it is inserted into L with support value 1. If an arrangement already exists in L , its frequency is increased by one. The complete set of frequent arrangements is determined by scanning the whole sequence and by increasing the support of

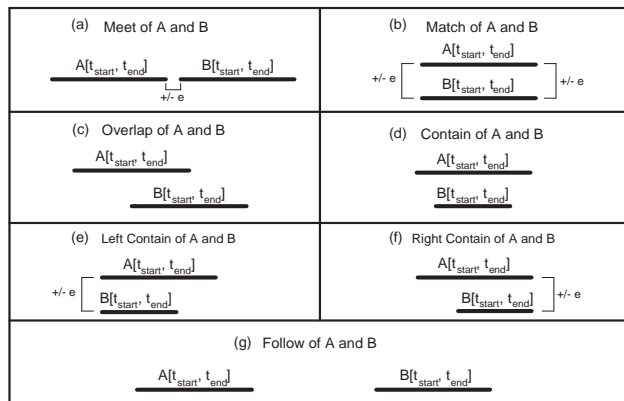


Fig. 4. Basic relations between two event-intervals: (a) Meet, (b) Match, (c) Overlap, (d) Contain, (e) Left-Contain, (f) Right-Contain, (g) Follow.

each arrangement by one, for every window in which it occurs. Eventually, the complete set of frequent arrangements of poly-regions in S is produced, by extracting those arrangements in L with support that satisfies the minimum support threshold.

C. Mining Frequent Arrangements in a Set of DNA Sequences

Let us now consider the case where we have multiple DNA sequences and want to extract the set of frequent arrangements of poly-regions that occur in the given set of sequences. We can consider each sequence as an e-sequence and the whole set as an *e-sequence database* [?]. Applying the Hybrid-DFS algorithm developed in [?] will yield the complete set of frequent arrangements of poly-regions. Notice that we could also apply spatial (i.e. place limits on the distance between two poly-regions) and structural constraints (i.e. apply regular expression constraints to the extracted patterns) during the mining process so as to focus on certain types of patterns. Various techniques and algorithms for this process are described in [?].

D. Complexity

The problem of discovering frequent arrangements of temporal intervals has exponential complexity with respect to the number of possible event labels. The enumeration tree and the pruning techniques used during the mining process decrease the cost significantly. However, depending on the nature of the input data, the set of event labels and the density of the e-sequences, the complexity can increase dramatically (in the worst case).

V. EXPERIMENTAL EVALUATION

Our experiments have three main targets: (1) analyze the performance of the proposed algorithms in terms of accuracy and runtime, (2) study the types of poly-regions that occur in different types of DNA regions (introns, exons, nucleosomes), and (3) study the frequent arrangements of poly-regions in these different region types. All the experiments have been performed on a 2.8Ghz Intel(R) Pentium(R) 4 dual-processor machine with 2.5 gigabytes main memory, running Linux with kernel 2.4.20. The algorithms have been implemented in C++, compiled using g++ along with the -O3 flag, and their runtime has been measured with the output turned off.

A. Datasets

Two different datasets have been used in our experimental evaluation. The first was taken from <http://www.ncbi.nlm.nih.gov>. This directory includes sequence records and map data generated at NCBI or used in NCBI resources. Sequence data include WGS supercontigs generated by the Broad Institute of MIT/Harvard and Agencourt Bioscience, along with RNAs and proteins generated through the NCBI Reference Sequence and NCBI Genome Annotation projects. The files in this directory provide assembled sequences for the chromosomes of the reference assembly. Runs of Ns are inserted into the sequence wherever there is a gap in the contig layout, e.g. between contigs, at the centromere, at the telomeres, or at large regions of heterochromatin. The NCBI Map Viewer (<http://www.ncbi.nlm.nih.gov/mapview/>) provides graphical views of the dog genome data. For our experiments we have used 39 chromosomes (including the *X* chromosome) of the organism *Canis familiaris* (dog). The second dataset was taken from <http://genie.weizmann.ac.il/pubs/nucleosomes06>. This directory includes the DNA sequences around all nucleosome regions of: the Yeast in vivo (119 nucleosomes) and in vitro (204 nucleosomes), the Chicken in vivo (177 nucleosomes) and the Mouse in vitro (87 nucleosomes) genomes with an explicit annotation of the nucleosome positions on the chromosomes. Before applying our algorithms, the input DNA sequences have been pre-processed to remove the runs of Ns.

B. Performance Analysis

The three proposed algorithms have been compared in terms of runtime and accuracy considering the following factors: (1) size of the input sequence, (2) density of the poly-regions, (3) size of the minimum and maximum windows.

Regarding runtime, the basic observation is that the third algorithm (majority vote-based) outperforms the rest. The sliding window approach is pretty fast, outperforming the recursive segmentation approach. In Figure ??, we show the performance of each algorithm with respect to the density constraint, which varies from 40% to 80%, for Chromosomes 1 (approximately 127 million bases) and *X* of the *Canis Familiaris* respectively. For Chromosome 1, the window range is [10, 20], whereas for Chromosome *X*, the window range is [20, 40].

Regarding accuracy, the sliding window approach achieved to find the complete set of poly-regions. The recursive segmentation was proved to be less accurate managing to find almost 80% (on average) of the total poly-regions. This was totally expected for both cases: the nature of the recursive segmentation is such that split points might be chosen inside some poly-regions. This can happen mainly at the first segmentations where the segments are relatively huge. As a result, these poly-regions are not going to be included in the final segmentation. In the case of the majority vote, the chosen window size might skip some poly-regions, due to its size and depending on the value of the density constraint. For example, let $S = \dots AACAA \dots$, $d = 80\%$, $w = 3$ and $max_win = 6$; due to the value of d , a poly-region will be reported only when all three literals in w are the same. Thus, the poly-region of literal *A* of size 5 shown in S will be skipped. The experimental evaluation however, showed that if the size of w is chosen to be $min_win/2$, the percentage of false negatives will be less than 11%. Table ?? presents some results regarding the accuracy of the algorithms showing that the majority vote method performs significantly better than the recursive segmentation.

C. Our Findings

In this section we present a study of the types of poly-regions identified in different regions of each chromosome. We consider three different types of DNA regions: (1) exons (coding regions), (2) introns (non-coding regions), (3) nucleosomes. Then, we give an analysis of the performance of the three algorithms regarding time and accuracy.

Chromosome	Arrangement	Support in %	Arrangement	Support in %
Chromosome 1	$\frac{A}{C}$	45%	$\frac{T}{G}$	46%
	$\frac{A+G}{T}$	36%	$\frac{C}{T}$ $\frac{A}{A}$	21%
Chromosome 2	$\frac{A}{C}$	43%	$\frac{C}{G}$	54%
	$\frac{G}{T}$	52%	$\frac{C+G}{T}$ $\frac{A}{A}$	13%
Chromosome X	$\frac{A+G}{T}$	37%	$\frac{C}{T}$ $\frac{A}{A}$	16%
	$\frac{C}{G}$ T	22%	$\frac{A+C}{C+G}$ T	11%

Fig. 5. A sample of the Extracted Set of Frequent Arrangements in Exons of Chromosomes 1, 2 and X of the Canis Familiaris. The poly-region size varied between 10 and 40 nucleotides.

Chromosome	Arrangement	Support in %	Arrangement	Support in %
Chromosome 1	$\frac{A}{TG}$	35%	$\frac{A}{T}$	66%
	$\frac{A+C}{C}$	21%	$\frac{C}{A}$ C	19%
Chromosome 2	$\frac{A}{T}$	25%	$\frac{T}{G}$	33%
	$\frac{G}{T}$	42%	$\frac{C+G}{T}$ $\frac{A}{A}$	26%
Chromosome X	$\frac{A+T}{C+G}$	33%	$\frac{C}{T}$ $\frac{A}{A}$	35%

Fig. 6. A sample of the Extracted Set of Frequent Arrangements in Introns regions of Chromosomes 1, 2 and X of the Canis Familiaris. The poly-region size varied between 10 and 40 nucleotides.

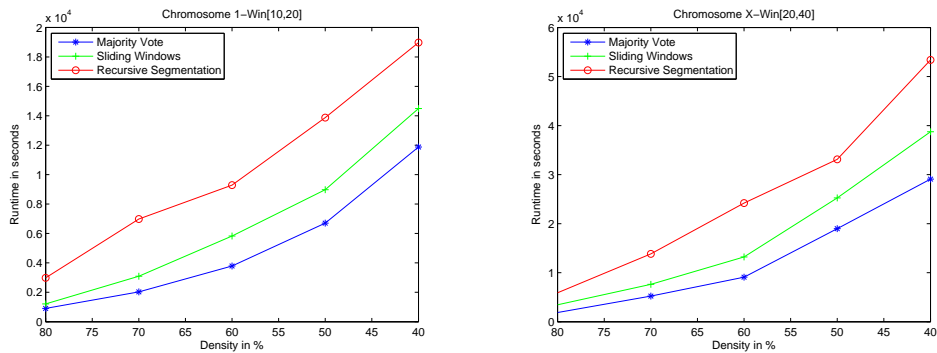


Fig. 7. Runtime comparison of the three algorithms for Chromosomes 1 (left Figure) and X(right Figure).

Arrangement	Support in %
$\begin{array}{c} \text{T} \\ \text{---} \\ \text{G} \\ \text{---} \end{array}$	36%
$\begin{array}{c} \text{T+G} \\ \text{---} \\ \text{A} \\ \text{---} \end{array}$	19%
$\begin{array}{c} \text{C+G} \\ \text{---} \\ \text{A} \\ \text{---} \end{array}$	14%
$\begin{array}{c} \text{T} \\ \text{---} \\ \text{G} \\ \text{---} \end{array}$	31%
$\begin{array}{c} \text{A+T} \\ \text{---} \\ \text{A+G} \\ \text{---} \end{array}$	7%
$\begin{array}{c} \text{TG} \\ \text{---} \\ \text{G+C} \\ \text{---} \end{array}$	11%
$\begin{array}{c} \text{T} \quad \text{T} \\ \text{---} \quad \text{---} \end{array}$	21%
$\begin{array}{c} \text{TG} \quad \text{TG} \\ \text{---} \quad \text{---} \end{array}$	24%

Fig. 8. A sample of the Extracted Set of Frequent Arrangements in nucleosome regions of the Yeast, Mouse and Chicken Genomes. The poly-region size varied between 10 and 40 nucleotides.

TABLE I
ACCURACY OF THE THREE ALGORITHMS FOR CHROMOSOMES 1, 38 AND X OF THE *Canis Familiaris*

Chromosome	Poly-Region size	Sliding Window	Recursive Segm	Majority Vote
1	[10, 20]	49325 (100%)	38223 (77%)	45582 (92%)
1	[18, 64]	26332 (100%)	23245 (88%)	24765 (94%)
38	[10, 20]	11285 (100%)	8195 (85%)	9980 (88%)
38	[18, 64]	8221 (100%)	6948 (72%)	7988 (97%)
X	[10, 20]	1793112 (100%)	1291762 (72%)	1605430 (89%)
X	[18, 64]	696261 (100%)	598455 (85%)	622987 (90%)

1) *Poly-regions in Exons and Introns*: The Dog Genome consists of 39 chromosomes out of which only 38 (the Y chromosome does not include exons, as expected) contain exons. In Table ?? we can see for some chromosomes, the number of introns and exons they contain, along with their minimum, maximum and average size. In Table ?? we can see the same statistics for the introns (non-coding regions). Tables ??, ?? and ?? show some statistics regarding the types of poly-regions discovered in coding regions of the Dog Genome for chromosomes 1, 20 and X. A much larger number of poly-regions have been discovered in non-coding regions and are shown in Tables ?? and ?? for chromosomes 1 and 20 respectively. Due to space limitations we present a sample of our findings. The minimum density constraint was set to 80% and the poly-region size varied between 10 and 60 nucleotide bases. We examined a total of 360457 exons with an average size between 147 and 186 nucleotides, and 194373 introns with an average size between 5096 and 27521 nucleotides. The main observation is that introns show a significantly larger accumulation of poly-regions than the exons, especially poly-As, poly-Cs, poly-Ts, poly-CTs and poly-TGs. On the other hand, exons have a high concentration of poly-As, poly-Ts and poly-TGs. Among all poly-regions of Type II with $\mathcal{S} = 3$, only poly-AATs, poly-ATTs, poly-TATs and poly-ATAs show a significant occurrence in exons whereas in introns we can also have poly-CCTs, poly-CTTS poly-GAAs and poly-GTTs.

2) *Poly-regions in Nucleosome Regions*: Our algorithms have also been applied to nucleosome regions of the Yeast (in vivo and in vitro), Chicken (in vivo) and Mouse (in vitro) genomes. The extracted poly-regions from the aforementioned genomes are shown in Tables ??, ??, ??, ?? respectively. The main observation is that nucleosome regions show a larger accumulation of poly-regions than exons; especially

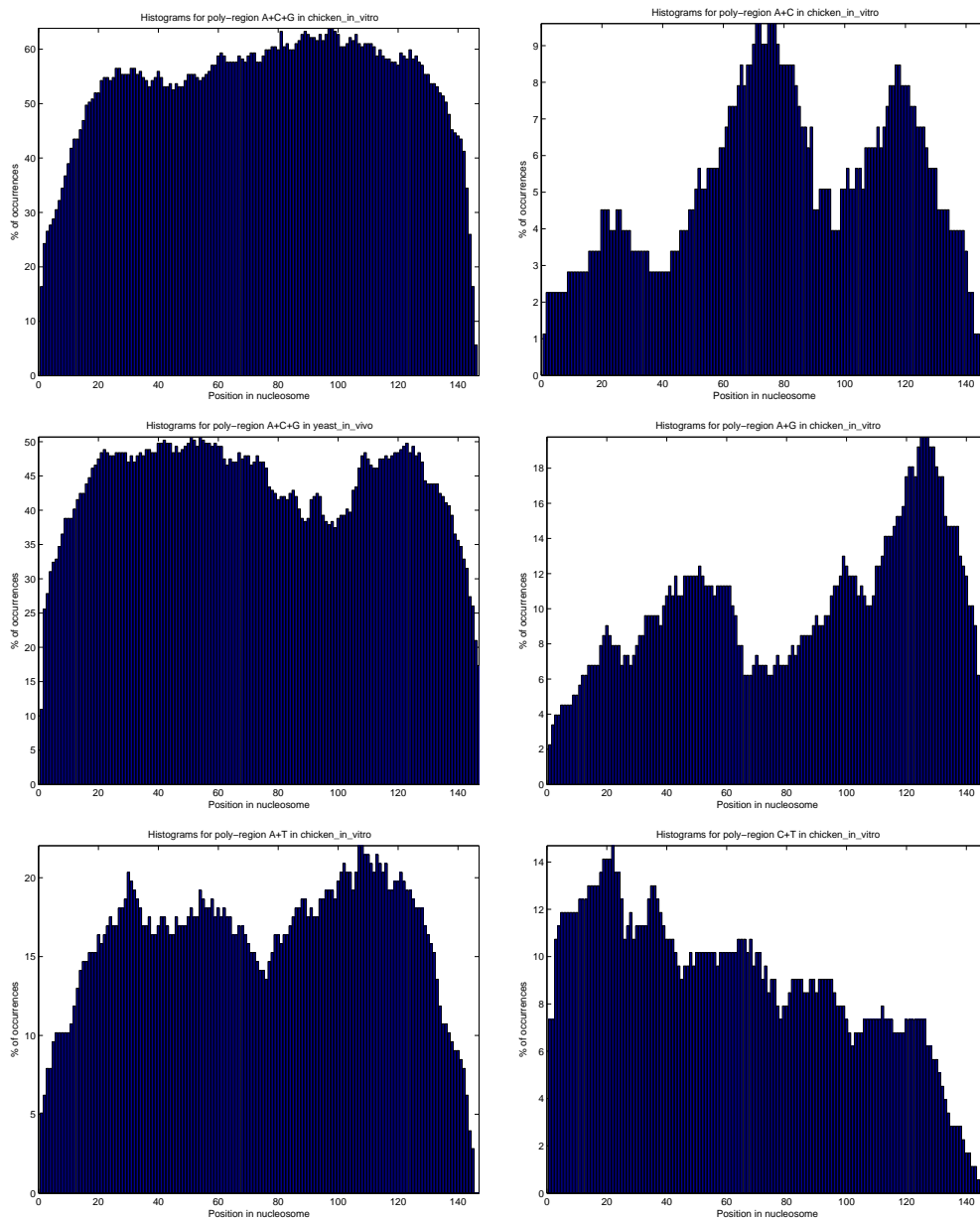


Fig. 9. Histograms for each different frequent poly-region type for the 4 organisms of our nucleosome dataset. The x-axis corresponds to the actual position on the nucleosome and the y-axis represents the percentage of nucleosomes where this poly-region occurs in that specific position.

poly-regions of Type I with $|\mathcal{I}| \geq 2$, are present in almost every nucleosome. We also noticed a high occurrence of poly-CAs and poly-TGAs in the Mouse in vitro genome, which is not true for the other genomes we examined.

Another interesting observation was that some poly-regions tend to appear in certain positions on the nucleosomes. For example there can be a higher concentration of some types of poly-regions at the beginning of each nucleosome or at the end, or we can even have some type of periodicity along each nucleosome. In Figures ?? and ?? we see the histograms for each different frequent poly-region type for the 4 organisms of our nucleosome dataset. The x-axis corresponds to the actual position on the nucleosome and the y-axis represents the percentage of nucleosomes where this poly-region occurs in that specific position. The basic observations regarding the poly-region positioning on nucleosomes include:

- 1) There is a high occurrence of poly-regions of Type I and size 2 (i.e. $A + Cs$, $C + Gs$, etc...) with a

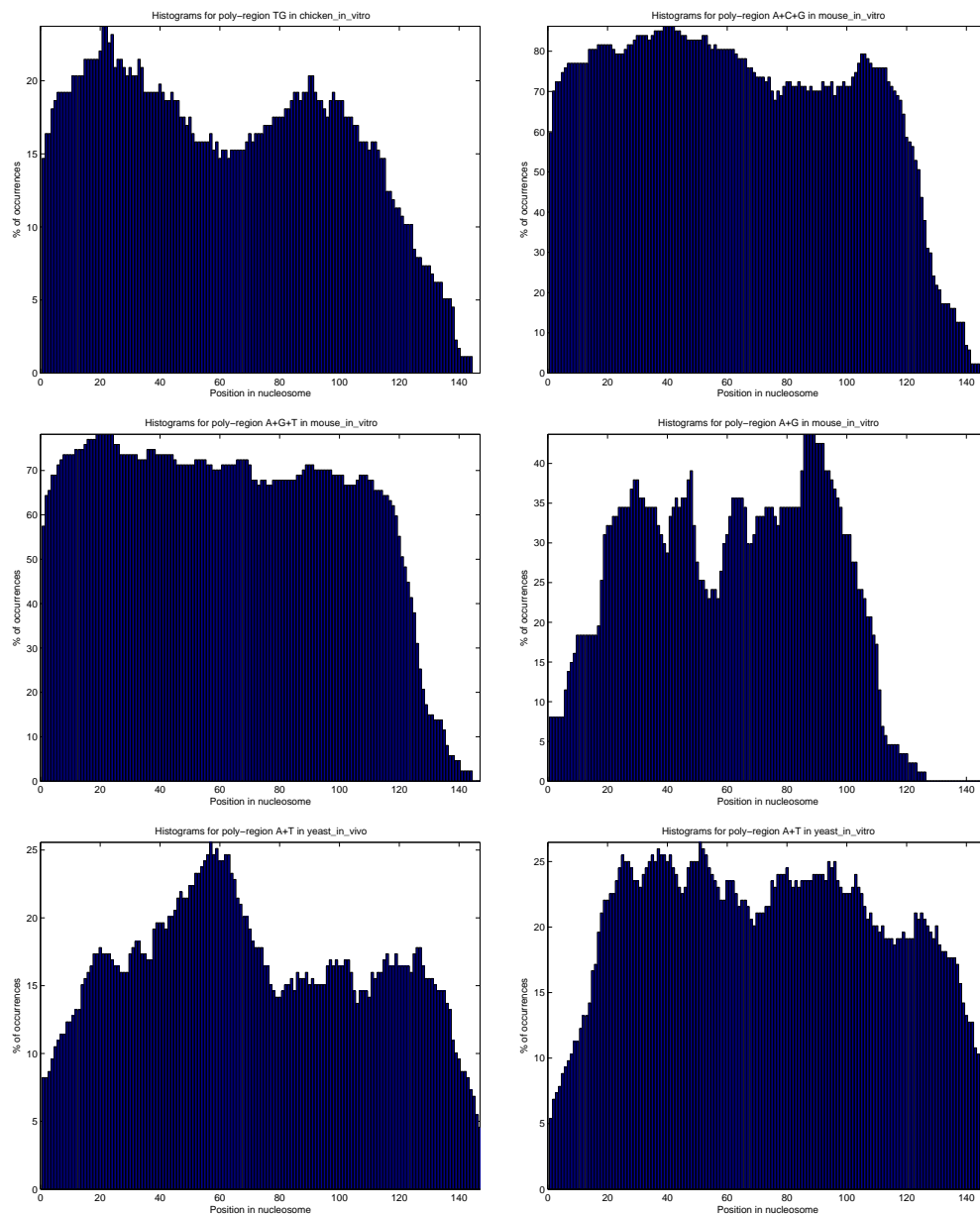


Fig. 10. Histograms for each different frequent poly-region type for the 4 organisms of our nucleosome dataset. The x-axis corresponds to the actual position on the nucleosome and the y-axis represents the percentage of nucleosomes where this poly-region occurs in that specific position.

frequency of approximately 20%. As for poly-regions of size 3, their positioning is kind of random, which is expected due to the fact that the alphabet size is only 4.

- 2) In some cases there is a sharp drop off towards the end of the nucleosomes. This is true especially for poly-regions of Type I and sizes 2 and 3: poly- $A + C + G$, poly- $A + G + T$ and poly- TG in Mouse in vitro, poly- $A + T$ in the chicken in vitro.
- 3) Some signs of periodicity are detected in a few histograms, for example in the poly- $C + G$ in the Yeast in vivo and poly- $A + C$ in the Chicken in vitro.
- 4) The only *poly-di-nucleotide region* (poly-region of Type II and size 2) that appears in nucleosomes is TG . In the Chicken in vivo genome, there is a high concentration of TGs at the beginning of the nucleosomes.

3) *Extracting Temporal Arrangements*: Finally, an efficient mining algorithm has been applied to the extracted poly-regions, as described in Section ??, to detect frequent temporal relations between them.

TABLE II
INTRONS AND EXONS IN THE DOG GENOME

Chromosome	Introns				Exons			
	# of Introns	Minimum Size	Maximum Size	Average Size	# of Exons	Minimum Size	Maximum Size	Average Size
X	7019	9	3226672	17885	15114	1	5862	174
1	10523	2	1425363	11412	16895	2	6544	169
5	10382	17	2337010	8401	20229	1	6491	157
10	6552	5	1957447	10443	11136	1	6435	151
15	5022	6	1633963	12569	9727	2	4647	155
20	9848	2	1819157	5757	21132	1	6642	157
25	4082	8	1252686	12465	7022	1	11480	161
30	4595	5	1247264	8558	9216	2	8158	150
35	1339	15	1184727	19488	1880	1	5960	186
38	1876	17	1732858	12529	3210	2	3457	162

TABLE III
TYPES OF POLY-REGIONS WITH DENSITY $\geq 80\%$ IN EXONS OF CHROMOSOME 1

Poly-Region Type	Percentile over all regions	Percentile among exons
A	0.42%	21.29%
T	0.45%	23.74%
TG	2.25%	56.69%
A+C	2.96%	75.19%
A+G	4.38%	79.29%
A+T	10.11%	83.65%
C+G	1.60%	36.26%
C+T	4.40%	77.87%
G+T	2.82%	74.46%
A+C+G	16.64%	94.38%
A+C+T	27.51%	96.70%
A+G+T	25.35%	96.90%

TABLE IV
TYPES OF POLY-REGIONS WITH DENSITY $\geq 80\%$ IN EXONS OF CHROMOSOME 20

Poly-Region Type	Percentile over all regions	Percentile among exons
A	0.38%	17.16%
T	0.44%	18.70%
TG	2.81%	64.38%
A+C	3.32%	77.63%
A+G	4.85%	80.42%
A+T	8.01%	73.71%
C+G	2.86%	49.31%
C+T	4.50%	78.75%
G+T	2.88%	75.29%
A+C+G	21.79%	95.51%
A+C+T	23.83%	95.10%
A+G+T	22.99%	95.54%

TABLE V
TYPES OF POLY-REGIONS WITH DENSITY $\geq 80\%$ IN EXONS OF CHROMOSOME X

Poly-Region Type	Percentile over all regions	Percentile among exons
A	0.42%	22.75%
T	0.41%	22.60%
TG	2.16%	58.09%
A+C	3.11%	75.51%
A+G	4.56%	79.24%
A+T	9.83%	87.71%
C+G	0.98%	32.07%
C+T	4.43%	77.09%
G+T	2.71%	73.33%
A+C+G	16.07%	94.06%
A+C+T	28.47%	97.57%
A+G+T	25.72%	97.61%

TABLE VI
TYPES OF POLY-REGIONS WITH DENSITY $\geq 80\%$ IN INTRONS OF CHROMOSOME 1

Poly-Region Type	Percentile over all regions	Percentile among introns
A	0.46%	69.48%
C	0.05%	38.27%
G	0.05%	38.74%
T	0.44%	69.80%
AC	0.02%	21.45%
AG	0.08%	46.55%
AT	0.05%	40.06%
CA	0.03%	32.38%
CT	0.08%	46.16%
GA	0.05%	40.98%
GT	0.02%	21.87%
TA	0.05%	36.11%
TC	0.05%	41.03%
TG	2.17%	90.41%
AAT	0.03%	35.17%
ATT	0.03%	35.38%
CCT	0.01%	18.89%
CTT	0.02%	21.72%
GAA	0.03%	22.82%
TAA	0.03%	32.23%
TTA	0.03%	32.46%
TTC	0.02%	22.40%
A+C	2.90%	95.52%
A+G	4.39%	96.27%
A+T	10.30%	96.78%
C+G	1.22%	77.26%
C+T	4.16%	95.45%
G+T	2.59%	94.75%
A+C+G	16.15%	99.11%
A+C+T	28.29%	99.57%
A+G+T	26.00%	99.66%

TABLE VII
TYPES OF POLY-REGIONS WITH DENSITY $\geq 80\%$ IN INTRONS OF CHROMOSOME 20

Poly-Region Type	Percentile over all regions	Percentile among introns
A	0.41%	53.04%
C	0.09%	40.80%
G	0.09%	39.97%
T	0.42%	53.48%
AG	0.09%	38.92%
AT	0.04%	22.86%
CA	0.04%	26.34%
CT	0.09%	40.13%
GA	0.06%	33.79%
TA	0.03%	20.10%
TC	0.06%	33.85%
TG	2.71%	89.34%
AAT	0.03%	25.53%
ATT	0.03%	25.21%
TAA	0.03%	24.00%
TTA	0.03%	23.68%
A+C	3.04%	93.06%
A+G	4.62%	93.29%
A+T	8.11%	88.64%
C+G	2.22%	82.81%
C+T	4.49%	93.53%
G+T	2.80%	92.29%
A+C+G	20.56%	99.01%
A+C+T	25.40%	98.42%
A+G+T	24.16%	98.73%

TABLE VIII
TYPES OF POLY-REGIONS WITH DENSITY $\geq 80\%$ IN THE YEAST IN VIVO

Poly-Region Type	Percentile over all regions	Percentile among nucleosomes
A	0.49%	23.62%
T	0.15%	14.07%
TG	1.79%	56.78%
A+C	2.85%	88.94%
A+G	3.92%	91.46%
A+T	7.75%	97.49%
C+G	0.46%	31.16%
C+T	3.20%	87.94%
G+T	2.69%	82.41%
A+C+G	14.38%	99.50%
A+C+T	31.17%	100.00%
A+G+T	30.64%	100.00%

TABLE IX
TYPES OF POLY-REGIONS WITH DENSITY $\geq 80\%$ IN THE YEAST IN VITRO

Poly-Region Type	Percentile over all regions	Percentile among nucleosomes
A	0.17%	14.22%
T	0.22%	20.10%
TG	2.09%	67.16%
A+C	2.82%	88.24%
A+G	3.49%	88.24%
A+T	8.34%	98.53%
C+G	0.32%	21.57%
C+T	3.27%	82.35%
G+T	2.53%	83.33%
A+C+G	13.39%	100.00%
A+C+T	32.94%	100.00%
A+G+T	30.04%	99.51%

TABLE X
TYPES OF POLY-REGIONS WITH DENSITY $\geq 80\%$ IN THE CHICKEN IN VIVO

Poly-Region Type	Percentile over all regions	Percentile among nucleosomes
A	0.25%	19.21%
T	0.23%	15.25%
TG	2.99%	72.32%
A+C	3.29%	85.88%
A+G	4.79%	90.40%
A+T	6.70%	88.14%
C+G	0.76%	31.64%
C+T	4.01%	87.01%
G+T	3.07%	84.75%
A+C+G	18.45%	98.87%
A+C+T	28.40%	99.44%
A+G+T	26.28%	100.00%

TABLE XI
TYPES OF POLY-REGIONS WITH DENSITY $\geq 80\%$ IN THE MOUSE IN VITRO

Poly-Region Type	Percentile over all regions	Percentile among nucleosomes
A	0.07%	9.20%
AC	0.33%	9.20%
CA	0.44%	11.49%
TG	2.47%	40.23%
A+C	4.90%	85.06%
A+G	10.04%	72.41%
A+T	2.91%	73.56%
C+G	0.76%	26.44%
C+T	0.88%	17.24%
G+T	2.43%	36.78%
A+C+G	29.78%	98.85%
A+C+T	17.91%	95.40%
A+G+T	26.44%	97.70%

Specifically, the algorithm has been applied to the poly-regions of Chromosomes 1, 2 and X of the *Canis Familiaris*. An interesting number of frequent patterns has been extracted. In all three cases we detected a great number of overlaps and contains between poly-As and poly-Ts (in exons) as well as poly-Cs and poly-Gs (in introns). Figure ?? gives a sample of the frequent arrangements that have been extracted from the introns of Chromosomes 1, 2 and X of the *Canis Familiaris*. Figure ?? shows the most interesting patterns in the same chromosomes but regarding coding regions. The most significant observation is the arrangement involving a follow relation of poly-TAs and poly- $\{C + G\}$ s. Similar but fewer arrangements have been discovered in the nucleosomes (of the Yeast, Mouse and Chicken genomes). A sample of the highest scoring arrangements is shown in Figure ??.

VI. CONCLUSION

We have formally defined the problem of detecting regions of high occurrence of a literal or set of literals in a sequence and proposed three efficient algorithms to solve it. The key idea of the first one is to use a set of sliding windows over the input sequence. Each sliding window keeps a set of statistics of a sequence segment that mainly includes the number of occurrences of each item in that segment. Combining these statistics efficiently yields the complete set of regions of high occurrence of the items of the given alphabet. The second algorithm applies an efficient segmentation technique that produces a set of segments. The third and more efficient approach applies a technique based on the majority vote achieving linear running time with a minimal number of false negatives. After identifying these regions, the sequence is converted to a sequence of labelled intervals (each one corresponding to a region). We further applied an efficient arrangement mining algorithm to extract the complete set of frequent temporal arrangements of the extracted regions we provided an extensive experimental evaluation of our algorithms by testing their efficiency on real DNA data. In our experiments, we extensively study the types of poly-regions and arrangements that occur frequently in different DNA regions (coding, non-coding, and nucleosomes).

Some directions for future research include: (1) improvement of our algorithms to be able to work efficiently in larger alphabet sizes, (2) application of our algorithms on proteins, and (3) application of the mining algorithm on multiple DNA and protein sequences aiming at the detection of arrangements of poly-regions that occur frequently among these sequences.

REFERENCES

- [1] Y.-X. Fu and R. N. Curnow, "Maximum likelihood estimation of multiple change points," *Biometrika*, vol. 77, pp. 563–573, 1990.
- [2] T. R. Bement and M. S. Waterman, "Locating maximum variance segments in sequential data," *Mathematical Geology*, vol. 9, pp. 55–61, 1977.
- [3] I. E. Auger and C. E. Lawrence, "Algorithms for the optimal identification of segment neighborhoods," *Bulletin of Mathematical Biology*, vol. 51, pp. 39–54, 1989.
- [4] G. A. Churchill, "Stochastic models for heterogeneous dna sequences," *Bulletin of Mathematical Biology*, vol. 51, no. 1, pp. 79–94, 1989.
- [5] —, "Hidden markov chains and the analysis of genome structure," *Computes and Chemistry*, vol. 16, no. 2, pp. 107–115, 1992.
- [6] J. W. Fickett, D. C. Torney, and D. R. Wolf, "Base compositional structure of genomes," *Genomics*, vol. 13, pp. 1056–1064, 1992.
- [7] R. Gwadera, A. Gionis, and H. Mannila, "Optimal segmentation using tree models," *icdm*, pp. 94–98, 2006.
- [8] V. E. Ramensky, V. Markeev, M. Roytberg, and V. Tumanyan, "Dna segmentation through the bayesian approach," vol. 7, pp. 215–231, 2000.
- [9] E. Carlstein, H. G. Mueller, and D. Siegmund, "Change-point problems," *Lecture Notes and Monograph Series*, vol. 23, no. 2, 1994.
- [10] J. V. Braun and H. G. Mueller, "Statistical methods for dna segmentation," *Statistical Science*, vol. 13, pp. 142–162, 1998.
- [11] J. V. Braun, R. K. Braun, and H. G. Mueller, "Multiple change-point fitting via quasi-likelihood, with application to dna sequence segmentation," *Biometrika*, vol. 87, pp. 301–314, 2000.
- [12] W. Szpankowski, W. Ren, and L. Szpankowski, "An optimal dna segmentation based on the mdl principle," in *IEEE Computer Society Bioinformatics Conference (CSB)*, 2003.
- [13] I. Grosse, P. V. Galvan, P. Carpena, R. R. Roldan, J. Oliver, and H. E. Stanley, "Analysis of symbolic sequences using the jensen-shannon divergence," *Physical Review E*, vol. 65, p. 041905, 2002.
- [14] C.-T. Zhang, F. Gao, and R. Zhang, "Segmentation algorithm for dna sequences," *Physical Review E*, vol. 72, p. 041917, 2005.
- [15] P. Bernaola-Galvan, R. Roman-Roldan, and J. L. Oliver, "Compositional segmentation and long-range fractal correlations in dna sequences," *Physical Review E*, vol. 53, pp. 5181–5189, 1996.

- [16] P. Bernaola-Galvan, P. Carpena, R. Roman-Roldan, and J. L. Oliver, "Mapping isochores by entropic segmentation of long genome sequences," 2001.
- [17] W. Li, P. Bernaola-Galvan, H. Fatameh, and I. Grosse, "Applications of recursive segmentation to the analysis of dna sequences," *Computes and Chemistry*, vol. 26, no. 2, pp. 491–510, 2002.
- [18] W. Li, "New stopping criteria for segmenting dna sequences," vol. 86, pp. 5815–5818, 2001.
- [19] J. Oliver, R. Romn-Roldn, J. Prez, and P. Bernaola-Galvn, "Segment: Identifying compositional domains in dna sequences," vol. 15, pp. 974–979, 1999.
- [20] J. C. Venter, "The sequence of the human genome," *Science*, vol. 291, pp. 1304–1351, 2001.
- [21] F. Larsen, G. Gundersen, R. Lopez, and H. Prydz, "Cpg islands as gene markers in the human genome," *Genomics*, vol. 13, pp. 1095–1107, 1992.
- [22] J. O. Pedro, "Isochore chromosome maps of the human genome," 2001.
- [23] E. Segal, Y. Fondufe-Mittendorf, L. Chen, A. Thstrm, Y. Field, I. K. Moore, J.-P. Z. Wang, and J. Widom, "A genomic code for nucleosome positioning," *Nature*, vol. 442, no. 7104, pp. 772–778, July 2006.
- [24] P. Papapetrou, G. Benson, and G. Kollios, "Discovering frequent poly-regions in dna," *icdm*, pp. 244–253, 2006.
- [25] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, "Discovering frequent arrangements of temporal intervals," in *Proc. of IEEE ICDM*, 2005, pp. 354–361.
- [26] H. Mannila and H. Toivonen, "Discovering generalized episodes using minimal occurrences," in *Proc. of ACM SIGKDD*, 1996, pp. 146–151.
- [27] J. Misra and D. Gries, "Finding repeated elements," *Sci. Comput. Program.*, vol. 2, no. 2, pp. 143–152, 1982.
- [28] L. Golab, D. DeHaan, E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro, "Identifying frequent items in sliding windows over on-line packet streams," in *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM Press, 2003, pp. 173–178.
- [29] J. Allen and G. Ferguson, "Actions and events in interval temporal logic," The University of Rochester, Tech. Rep. 521, July 1994.
- [30] C. Freksa, "Temporal reasoning based on semi-intervals," *Artificial Intelligence*, vol. 54, no. 1, pp. 199–227, 1992.
- [31] P. Papapetrou, "Constraint-based mining of frequent arrangements of temporal intervals," Master Thesis: Department of Computer Science, Boston University, Tech. Rep., January 2007.