Boston University Computer Science Technical Report No. 2006-027, October 15, 2006

Discovering Frequent Poly-Regions in DNA Sequences

Panagiotis Papapetrou[†] Gary Benson[††] George Kollios[†]
[†]Department of Computer Science, [††]Departments of Biology and Computer Science
Boston University
panagpap@cs.bu.edu, gbenson@bu.edu, gkollios@cs.bu.edu

Abstract

The problem of discovering arrangements of regions of high occurrence of one or more items of a given alphabet in a sequence, is studied, and two efficient approaches are proposed to solve it. The first approach is entropy-based and uses an existing recursive segmentation technique to split the input sequence into a set of homogeneous segments. The key idea of the second approach is to use a set of sliding windows over the 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. After identifying these regions, the sequence is converted to a sequence of labeled intervals (each one corresponding to a region). An efficient algorithm for mining frequent arrangements of temporal intervals on a single sequence is applied on the converted sequence to discover frequently occurring arrangements of these regions. The proposed algorithms are tested on various DNA sequences producing results with potentially significant biological meaning.

1 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 specically 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 [12]. For the same problem, a dynamic programming approach has been introduced in [3] that computes the global maximum, whereas [2] proposed an extension where there is no restriction on the segment size, (2) the hidden Markov chain model. Specifically, [8, 9] 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, (2) the walking Markov model, which is a continuously varying stochastic process. [10] 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.

Simultaneously, there have been studies on similar problems, called “change-point problems” that have been applied to DNA sequence segmentation [7, 6, 5]. 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.

Another family of DNA segmentation algorithms includes those that work in a hierarchical manner (top-to-bottom). In particular, they employ a 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 [13, 19]. Such algorithms have been proposed in [4, 13, 19] 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 [15] 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.

Last but not least, a sliding window approach with fixed size window has been applied on the human genome [18, 14] to detect $G + C$-rich regions and CpG islands.

To the best of our knowledge, all current approaches target specific compositions (mainly $G + C$-rich or CpG islands). Furthermore, there have been no studies on temporal relations that may occur between these regions. In this paper, we propose two general approaches to finding any type of “poly”-regions in DNA, and apply an efficient mining algorithm to extract frequent patterns between those regions.

In this paper we make the following contributions: (1) we formally define the problem of detecting regions of high occurrence of a literal or set of literals in a sequence, (2) we propose two efficient algorithms to solve the problem, (3) we further apply an efficient arrangement mining algorithm to extract the complete set of frequent temporal arrangements of these regions, (4) we provide experimental evaluation of our algorithms by testing their efficiency on the dog genome.

2 Problem Formulation

A *sequence* $\mathcal{S} = \{s_1,\ s_2,\ \dots\ ,\ s_m\}$ is an ordered list of items, where each s_i belongs to an alphabet Σ. In the case of DNA sequence, each s_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 *High region* or H*-region* is a segment of S, where there is a “high occurrence” of one or more items of Σ. Let $H^{d,k} = \{\mathcal{I},\ p_{start},\ p_{end}\}$ denote an H-region of k items with density d, that starts at item $s_{p_{start}}$ and ends at item $s_{p_{end}}$. $\mathcal{I}$ is a set of items that works as a label for the H-region and is determined by the k dense items in the H-region. Also, $s_{p_{start}} \in \mathcal{I}$ and $s_{p_{end}} \in \mathcal{I}$. An H-region $H^{d,k}$ has density d, if each of the k items occurs in the region with a frequency of at least d/k %. d is the minimum % in the segment of the items in $\mathcal{I}$. Given a density threshold $min\_density$, an H-region $H^{d,k}$ is *dense* if $d \geq min\_density$. Consider for example the two H-regions in Figure 1; (1) $H^{80,1} = \{\{A\},\ 5,\ 14\}$: in this case we have a region of “high occurrence” of nucleotide A with density 80%, (2) $H^{80,2} = \{\{A,\ C\},\ 20,\ 29\}$: in this case we have a region of “high occurrence” of nucleotides A and C, where each one has a density of 40%. Notice that the density is meaningful for small values of k. In the case of DNA, k should either be 1 or 2, i.e. if a region has a high occurrence of all four nucleotides (or of three) then the occurrences are rather random and have no particular meaning. Note that the terms ‘‘*poly”-region* and H*-region* are synonyms and are used interchangeably in this paper.

Given two H-regions $H_1^{d,k} = \{\mathcal{I}^1,\ p^1_{start},\ p^1_{end}\}$, $H_2^{d,k} = \{\mathcal{I}^2,\ p^2_{start},\ p^2_{end}\}$, the *merging* of $H_1^{d,k}$ and $H_2^{d,k}$ is a new H-region $H_{12}^{d,k} = \{\mathcal{I}^{12},\ p^{12}_{start},\ p^{12}_{end}\}$, with $p^{12}_{start} = min\{p^1_{start}, p^2_{start}\}$, $p^{12}_{end} = max\{p^1_{end}, p^2_{end}\}$, and $\mathcal{I}^{12} = \mathcal{I}^1 = \mathcal{I}^2$. Notice that merging is only allowed when $I^1 = I^2$ Also, an H-region $H_1^{d,k} = \{\mathcal{I}^1,\ p^1_{start},\ p^1_{end}\}$ is said to be contained in another H-region $H_2^{d,k} = \{\mathcal{I}^2,\ p^2_{start},\ p^2_{end}\}$, if $p^2_{start} \leq p^1_{start}$, $p^2_{end} \geq p^1_{end}$, and $\mathcal{I}^1 = \mathcal{I}^2$. A dense H-region $H_1^{d_1,k}$ is *maximal*, if there exists no H-region $H_2^{d_2,k}$ such that $d_2 \geq min\_density$ and $H_1^{d_1,k}$ is contained in $H_2^{d_2,k}$.

An H-region can be seen as an *event interval*, which (based on [17]) is a triple $(e_i,\ t^i_{start},\ t^i_{end})$, where e_i is an event label, t^i_{start} is the event start time and t^i_{end} is the end time. In our case, the start and end times correspond to the start and end positions of each H-region on 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 H-regions of a DNA sequence S constitutes the corresponding e-sequence of S. A more detailed analysis on the above terminology and concepts is given in Section 4.1.

Our goal is to first find the complete set of maximal H-regions given an input sequence and then apply an efficient algorithm for mining frequent arrangements of temporal intervals [17] to discover arrangements of H-regions that occur frequently in that sequence. However, in [17] the input

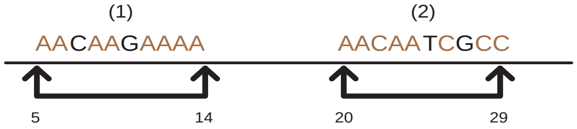


Figure 1. Example of two H-regions.

to the mining algorithm is a set of e-sequences, whereas in our case it is a single e-sequence. Thus, an approach similar to that described in [16] for mining frequent episodes over a sequence of instantaneous events can be employed.

Problem Statement: Given a sequence $S = \{s_1, s_2, ..., s_m\}$, a density constraint d, a minimum window size min_win, a maximum window size max_win and a support threshold min_sup, we want: (1) to discover the complete set $\mathcal{H}_S$ of maximal H-regions in S, where each region $H^{d,k} = \{\mathcal{I},\ p_{start},\ p_{end}\}$ has density of at least d and size $|H^{d,k}| = p_{end} - p_{start} + 1\ \in [min\_win, max\_win]$, and then (2) given $\mathcal{H}_S$ and a support threshold min_sup, extract the complete set $\mathcal{F}$ of frequent arrangements of H-regions in $\mathcal{H}_S$.

3 Extracting H-regions

In this Section we present two approaches for extracting the set of H-regions in a sequence of items that belong to a given alphabet. 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, whereas the second one implements a set of sliding windows over the sequence.

3.1 Recursive Segmentation

One approach for segmenting a sequence into H-regions is described in this section. The idea of recursive segmentation based on a measure of divergence has been used in earlier works, [4, 13, 19], and [15] 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 any type of "poly"-region. The main difference in our approach is that the recursive segmentation does not use the standard s_0 [4] 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) [19], the Jensen-Shannon Divergence (JSD) [13], 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 H-regions. Through a sequential scan of each segment these regions are identified by checking whether there exist a literal or set of literals (two literals in our case) that satisfy the density constraint in that segment.

Before proceeding to a detailed description of the algorithm, let as first give some basic definitions. Let $S = \{s_1,\ s_2,\ ...\ ,s_m\}$ 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 $S_g = \{n_1,\ n_2,\ ...\ ,n_{M-1}\}$, 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, $f(P) = \{f_i,\ i = 1, ..., t\}$ denotes the set of frequencies of each item in $P(S)$, where $f_i = \frac{number\ of\ occurrences\ of\ item\ i\ in\ S}{|S|}$, and t is the number of distinct items.

Let $H = -\sum f_i log_2 f_i,\ for\ i = 1, ..., t$ 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 $D(n) = H - (\frac{n}{m}H_{left} + \frac{m-n}{m}H_{right})$ (1), 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 H-regions.

3.1.1 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 that max_win, the recursion stops and P is reported, since it is a candidate H-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.

```
input  : $S$: the input sequence.
         $\Sigma$: the alphabet.
         $min\_win$: the minimum window size.
         $max\_win$: the maximum window size.
         $d$: the density constraint.
output: $\mathcal{H}_S$: the set of $H$-regions extracted from
         sequence $S$.
// Initialization Phase
$S = remove\_noise(S)$ // removes $N$s from $S$.
$K = \frac{(|\Sigma|+1)|\Sigma|}{2}$ // K: number of runs.
$H = \emptyset$; // H: the set of $H$-regions.
for $i = 0 : i < K : i{+}{+}$ do
    $p = find\_split(S, JSD)$;
    // finds split point $p$ where JSD
    // is maximized.
    if $|S[0, p]| > max\_win$ then
        $Algorithm1(S[0, p])$;
        // apply the algorithm on $S[0, p]$.
    end
    else
        $M = insert\_segment(S[0, p])$;
        // segment is inserted into $M$.
    end
    if $|S[p + 1, |S||| > max\_win$ then
        $Algorithm1(S[p + 1, |S|])$;
        // apply the algorithm on $S[p + 1, |S|]$.
    end
    else
        $M = insert\_segment(S[0, p])$;
        // segment is inserted into $M$.
    end
end
foreach $s \in M$ do
    if $s$ corresponds to an $H$-region then
        $insert\_to\_H(s)$; // $s$ is inserted into $H$
    end
end
```

Algorithm 1: *The recursive segmentation approach.*

To improve the efficiency of the segmentation we apply a preprocessing step, which has been suggested in [15] for the detection of isochores. Thus, when looking for H-regions of two nucleotides, say nucleotide Y and Z, the original sequence is transformed to a new sequence that consists of only three types of literals: those that correspond to nucleotides Y and Z, and those that do not (represented by literal X). For example, if $S = ACAAAGCGA$ and we are looking for H-regions of A and G, S will be converted to $S' = AXAAAGXGA$. The benefit of this replacement is the following: if one region is of high occurrence of two nucleotides and in the other region (the rest of the subsequence under consideration) all nucleotides of the different type are represented by one literal, and thus its entropy will definitely be larger. Also notice that the above replacement improves the runtime of the algorithm. This is expected, since the alphabet size has been reduced, achieving faster entropy calculations.

The steps of the extended algorithm for the case of H-regions of two nucleotides are given below:

1. Given an input sequence S, for each combination of two literals in Σ, convert S to S' as described above.

2. Given S', calculate $JSD(P, Q)$, 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 H-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 an H-region is found it is reported.

A brief pseudo-code for this approach is given in Algorithm 1. The same approach is followed for H-regions of one nucleotide.

3.1.2 Complexity

Every time the sequence is split into two subsequences. The number of splits is $O(log(m/(max\_win - min\_win)))$, where N 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 $O(mlogm)$. Now, given that the alphabet size is Σ, the number of times the algorithm is run is $K = \frac{(|\Sigma|+1)|\Sigma|}{2}$. Thus, the total runtime of the algorithm is $O(Kmlogm)$, and since K is a constant, this becomes $O(mlogm)$.

3.2 Sliding Windows

The key idea behind this approach is to use a set of sliding windows over the input sequence. Each sliding window will keep statistics of a segment that will mainly include the number of occurrences of each alphabet item in that segment. Combining these statistics efficiently produces the complete set of H-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 $\mathcal{W}$. 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 triple $\{\mathcal{C}^i,\ w^i_{start},\ w^i_{end}\}$, where $\mathcal{C}^i$ 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. $\mathcal{C}^i$ is a set of t counters $\{C_1,\ C_2,\ \dots,\ C_t\}$ one for each item in Σ. The value of each counter is the number of occurrences of the corresponding item in the window. Moreover, the piece of S covered by $\mathcal{W}$ is stored at each time instance. 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 H-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 in a window w^i is found to satisfy the density constraint, then w^i is reported as an H-region. In parallel, we keep a list L of the H-regions discovered so far. Each record in L corresponds to an H-region label and points to a list of all the H-regions discovered so far with this label. Upon discovery of a new H-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.

3.2.1 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 H-region into L the Merging Phase is activated, to identify any old H-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 $C^{i-1}_j,\ for\ j = 1, \dots, t$ denote the counters of the (i-1)-th window. Then $C^i_j = C^{i-1}_j,\ for\ j = 1, \dots, t$. 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 an H-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 characters that satisfy the density constraint. Starting with the window of maximum size, if item c is found to satisfy the density constraint, then this window is reported as an H-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 H-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 items of type c 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 items (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 H-region for each item. Consider Figure 1(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 an H-region of nucleotide C, we should skip w^9.

The above method produces a set $\mathcal{H}_S$ of H-regions for the input sequence $\mathcal{S}$. A brief pseudo-code for this approach is given in Algorithm 3.

3.2.2 Complexity

In this section we give the time and space complexity of our algorithm. 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 $|\Sigma|$ counters, one for each literal, which yields a total of $|\Sigma|M$ 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 $O(|\Sigma|M + max\_win)$. Now, let the input sequence be of size $m = |S|$. 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 k counters is checked and the last element of each window is removed. Therefore, for each slide a total of Mk counters are accessed. Also, when a window is determined to constitute an H-region, at most M records are accessed in the list L to check whether it overlaps with an existing H-regions. The above analysis yields a time complexity of $O(mM)$.

4 Discovering Frequent Arrangements of H-Regions in a DNA Sequence

In this Section we apply an efficient algorithm [17] for mining frequent arrangements of H-regions on a single input sequence of event intervals.

4.1 Background

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

```
input  : $S$: the input sequence.
         $\Sigma$: the alphabet.
         $min\_win$: the minimum window size.
         $max\_win$: the maximum window size.
         $d$: the density constraint.
output: $\mathcal{H}_S$: the set of $H$-regions extracted from
         sequence $S$.
// Initialization Phase
$S = remove\_noise(S)$ // removes $N$s from $S$.
for $i = 1 : i \leq max\_win - min\_win + 1 : i{+}{+}$ do
    $create\_windows(min\_win, max\_win)$;
end
// creates the set $\mathcal{W}$ of windows of size
// $[min\_win, max\_win]$. All windows are placed
// with their end point at position $max\_win$ of $S$.
// All counters have been updated so far.
$H = \emptyset$; // H: the set of $H$-regions.
for $k = max\_win + 1 : k \leq sizeof(S) : k{+}{+}$ do
    $slide()$;
    // slides the set of windows $\mathcal{W}$
    // one position to the right.
    $update\_counters()$;
    // updates counters in each window in $\mathcal{W}$.
    // starting from the window of size $max\_win$:
    $apply\_pruning()$;
    // pruning heuristic is applied
    // as described in Section 3.2
    foreach window $w \in \mathcal{W}$ do
        if exists a counter $C_j^w : \frac{C_j^w}{|w|} \geq d$ then
            $insert\_to\_H(w)$; // $w$ is inserted into $H$
        end
    end
end
```

Algorithm 2: *The sliding window approach.*

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 = $\{R\,(E_1,\ E_2),\ R\,(E_1,\ E_3),\ ...\ ,\ R\,(E_1,\ E_n),\ R\,(E_2,\ E_3),\ R\,(E_2,\ E_4),\ ...\ ,\ R\,(E_2,\ E_n),\ R\,(E_{n-1},\ E_n)\}$. R is the set of relations between each pair $(E_i,\ E_j)$, for $i = 1,\ ...,\ n$ and $j = i + 1,\ ...\ , n$, 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. Extending [17], we consider seven types of relations between two event intervals. 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 [1, 11].

Consider two event-intervals A and B, and assume that the user specifies a parameter ϵ used to define more flexible matchings between two points. The following relations are studied (see also Figure 2):

- **Meet (A, B):** In this case, B follows A, with B starting at the position where A terminates, i.e. $t_e(A) = t_s(B) \pm \epsilon$. This case is denoted as $A \sim B$ and we say that $A\ meets\ B$.

- **Match (A, B):** In this case, A and B are parallel, beginning and ending at the same point, i.e. $t_{start}(A) = t_{start}(B) \pm \epsilon$ and $t_{end}(A) = t_{end}(B) \pm \epsilon$. This case is denoted as $A||B$ and we say that $A\ matches\ B$.

- **Overlap (A, B):** In this case, the start point of B occurs after the start point of A, and A terminates before B, i.e. $t_{start}(A) < t_{start}(B)$, $t_{end}(A) < t_{end}(B)$ and $t_{start}(B) < t_{end}(A)$. This case is denoted as $A|B$ and we say that $A\ overlaps\ B$.

- **Contain (A, B):** In this case, the start point of B follows the start point of A and the termination of A occurs after the termination of B, i.e. $t_{start}(A) < t_{start}(B)\ and\ t_{end}(A) > t_{end}(B)$. This case is denoted as $A > B$ and we say that A *contains* B.

- **Left-Contain (A, B):** In this case, A and B start at the same point and A terminates after B, i.e. $t_{start}(A) = t_{start}(B) \pm \epsilon\ and\ t_{end}(A) > t_{end}(B)$. This case is denoted as $A \mid > B$ and we say that $A\ left\text{-}contains\ B$.

- **Right-Contain (A, B):** In this case, A and B end at the same position and the start point of A precedes that of B, i.e. $t_{start}(A) < t_{start}(B)\ and\ t_{end}(A) = t_{end}(B)\ \pm \epsilon$. This case is denoted as $A > \mid B$ and we say that $A\ right\text{-}contains\ B$.

- **Follow (A, B):** In this case, B occurs after A terminates, i.e. $t_{end}(A) \pm \epsilon < t_{start}(B)$. This case is denoted as $A \rightarrow B$ and we say that $B\ follows\ A$.

4.2 The Algorithm in Detail

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 H-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 $\mathcal{W} = |S| + win - 1$ 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: $freq(\mathcal{A}, win) = \frac{|\{w|\mathcal{A}\ occurs\ in\ w\}|}{|\mathcal{W}|}$.

The algorithm uses the *arrangement enumeration tree* structure, introduced in [17], 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 each arrangement by one, for every window in which it occurs. Eventually, the complete set of frequent arrangements of H-regions in S is produced, by extracting those arrangements in L with support that satisfies the minimum support threshold.

4.3 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).

5 Experimental Evaluation

In this section we present our experimental results on the performance of the proposed algorithms with respect to accuracy and runtime. 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.

5.1 Datasets

Here, we give a brief description of the datasets used for the experimental evaluation. The datasets we taken from $http://www.ncbi.nlm.nih.gov$. This directory includes sequence records and map data generated at NCBI or used

```
input  : $S$: the input e-sequence.
         $win$: the size of the sliding window.
         $min\_sup$: the support threshold.
output: $\mathcal{F}$: the set of frequent arrangements in $S$.
// Initialization Phase:
// $\mathcal{W}$ denotes the sliding window.
// $E_i$ is the event interval at position $i$ in $S$.
$\mathcal{W}.start = E_1$;
$\mathcal{W}.end = E_{win}$;
while $i \neq |S|$ do
    // apply the e-sequence enumeration technique
    // to extract frequent arrangements in $\mathcal{W}$.
    $\mathcal{C} = Extract\_patterns(\mathcal{W})$;
    $Update\_L(\mathcal{C})$;
    for $i = 1 : i \leq |\mathcal{L}| : i{+}{+}$ do
        if $\mathcal{L}_i \geq min\_sup$ then
            $\mathcal{F} = \mathcal{F} \cup \mathcal{L}_i$;
        end
    end
    $i$++;
end
```

Algorithm 3: *Mining frequent arrangements in a single e-sequence.*

in NCBI resources. 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 ($http : //www.ncbi.nlm.nih.gov/mapview/map\_search.cgi ?taxid = 9615$) provides graphical views of the dog genome data.

For our experiments, 39 chromosomes (including the X chromosome) of the organism $Canis\ familiaris$ (dog) have been used. Before applying our algorithms, the input DNA sequences have been pre-processed to remove the runs of Ns. Eventually each sequence followed alphabet $\Sigma = \{A, C, G, T\}$. The experimental evaluation is divided into two phases. In the first phase, our algorithms have been applied to the DNA sequences (chromosomes) and have been compared in terms of runtime and accuracy. In the second phase, we applied the mining algorithm described in Section 3 to extract frequent arrangements of H-regions on 3 chromosomes (1, 2 and 38).

5.2 Extracting H-regions

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

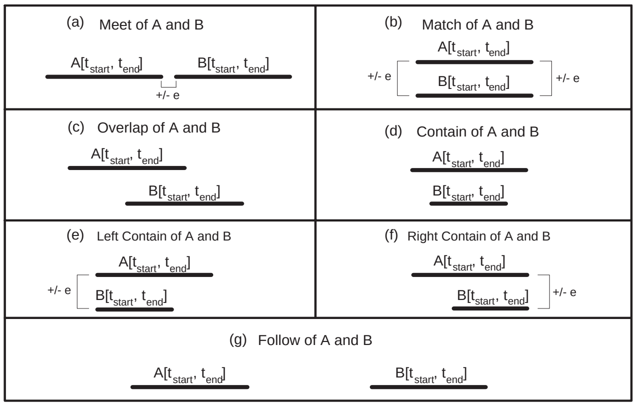
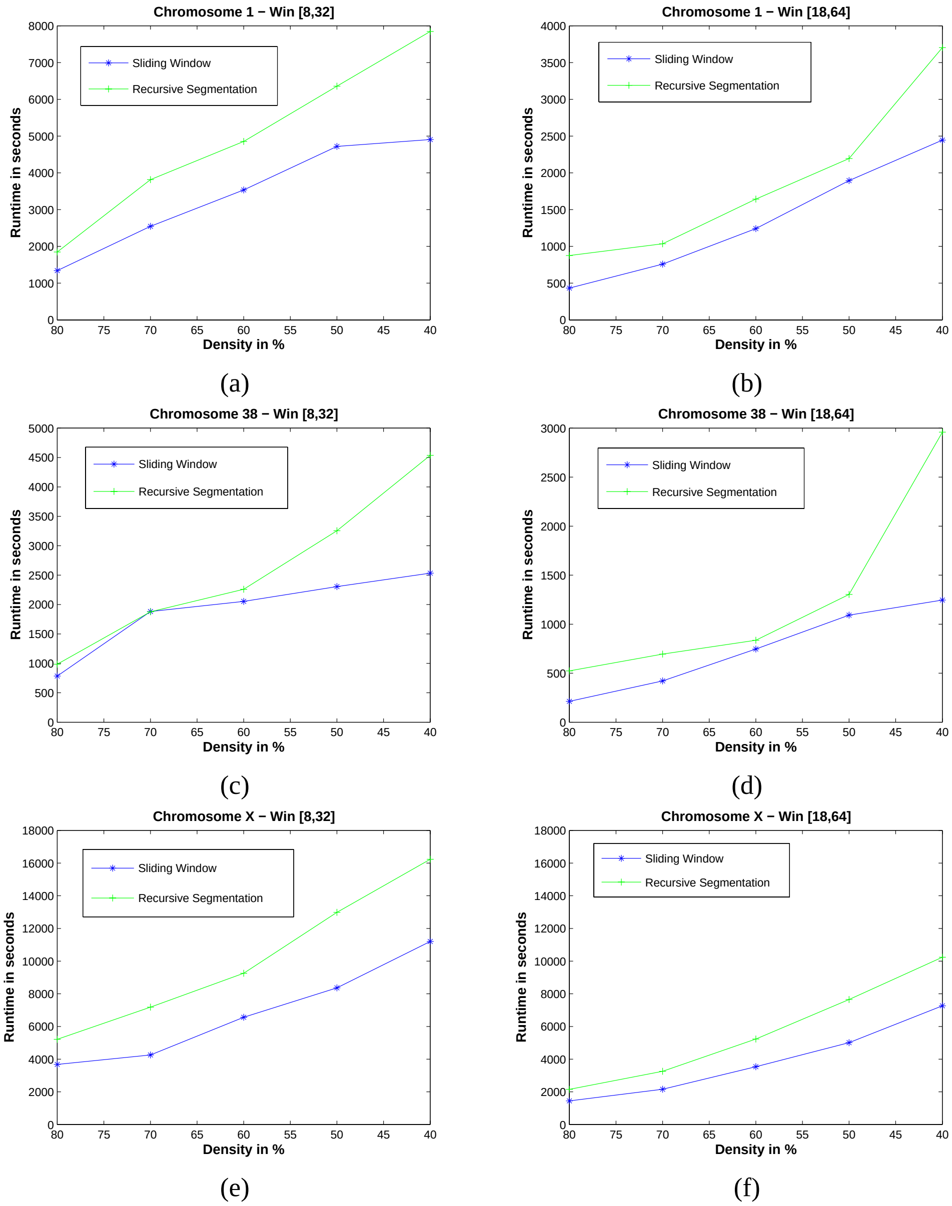


Figure 2. Basic relations between two event-intervals: (a) Meet, (b) Match, (c) Overlap, (d) Contain, (e) Left-Contain, (f) Right-Contain, (g) Follow.

Regarding runtime, the basic observation was that the sliding window approach outperformed the recursive segmentation approach both in small and large window ranges and density values. In Figure 3, we show the performance of each algorithm with regard to the density factor, which has been varied from 40% to 80%, on Chromosomes 1 (approximately 127 million bases), 48 (approximately 48 million bases) and X (approximately 412 million bases) of the $Canis\ Familiaris$ respectively. In Figures 3(a), 3(c) and 3(e) the window range is $[8, 32]$, whereas in Figures 3(b), 3(d) and 3(f) the window range is $[18, 64]$.

Regarding accuracy, the sliding window approach finds the complete set of H-regions because it does exhaustive search. The recursive segmentation approach had poorer performance but did not drop below 85% in accuracy. This was expected, because the recursive segmentation, might choose split points inside some H-regions. This can happen mainly at the early segmentations when the segments are large. Figure 5 shows some results regarding the accuracy of our algorithms. It can be seen that the accuracy of the recursive segmentation varies between 85% and 90%, performing slightly better in small sequences (Chromosome 38) and slightly worse in larger sequences (Chromosomes 1 and X). We also tried the recursive segmentation without the sequence conversion described in Section 3.1 and in Figure 5 we can see a significant difference between the two approaches. Each cell in the Figure shows the number of extracted H-regions and the percentage in the last two columns expresses the accuracy.

Also, Figure 6 gives an overview of how the number of the extracted H-regions increases as the density constraint decreases, showing that the smaller the window size, the more H-regions we get.



Chromosome 1 − Win [8,32]

(a)

Chromosome 1 − Win [18,64]

(b)

Chromosome 38 − Win [8,32]

(c)

Chromosome 38 − Win [18,64]

(d)

Chromosome X − Win [8,32]

(e)

Chromosome X − Win [18,64]

(f)

Figure 3. Results on Runtime Comparison: (a)Runtime Performance of the Two Algorithms for Chromosome 1 of the Canis Familiaris, with window range [8,32].; (b)Runtime Performance of the Two Algorithms for Chromosome 1 of the Canis Familiaris, with window range [18,64].; (c)Runtime Performance of the Two Algorithms for Chromosome 38 of the Canis Familiaris, with window range [8,32].;(d)Runtime Performance of the Two Algorithms for Chromosome 38 of the Canis Familiaris, with window range [18,64].;(e)Runtime Performance of the Two Algorithms for Chromosome X of the Canis Familiaris, with window range [8,32].;(f)Runtime Performance of the Two Algorithms for Chromosome X of the Canis Familiaris, with window range [18,64].

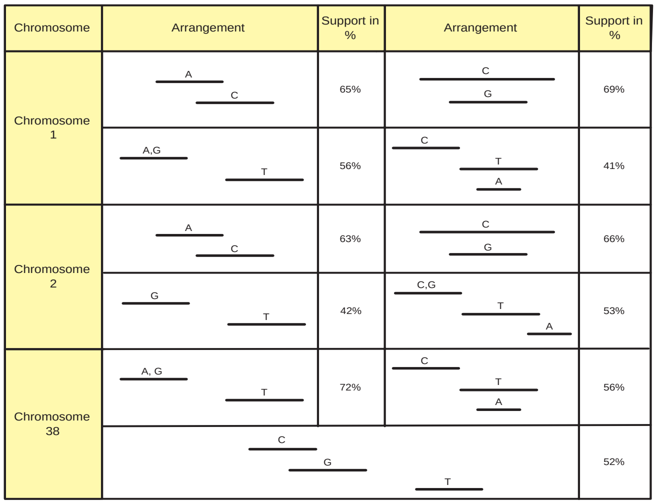


Figure 4. A sample of the Extracted Set of Frequent Arrangements for Chromosomes 1, 2 and 38 of the Canis Familiaris.

	Sliding Window	Recursive Segmentation (extended)	Recursive Segmentation (original)
Chromosome 1 (~127 million bs) win_size: [8,32]	53.563	48.287 (90%)	35.152 (65%)
Chromosome 1 (~127 million bs) win_size: [18,64]	26.332	23.245 (88%)	17.670 (67%)
Chromosome 38 (~48 million bs) win_size: [8,32]	102.482	87.433 (85%)	61.781 (60%)
Chromosome 38 (~48 million bs) win_size: [18,64]	78.221	69.475 (89%)	57.520 (73%)
Chromosome X(~412 million bs) win_size: [8,32]	1.873.670	1.594.091 (85%)	1.010.221 (54%)
Chromosome X(~412 million bs) win_size: [18,64]	696.261	598.455 (86%)	452.098 (65%)

Figure 5. Accuracy Performance of the Two Algorithms for Chromosomes 1, 38 and X of the Canis Familiaris.

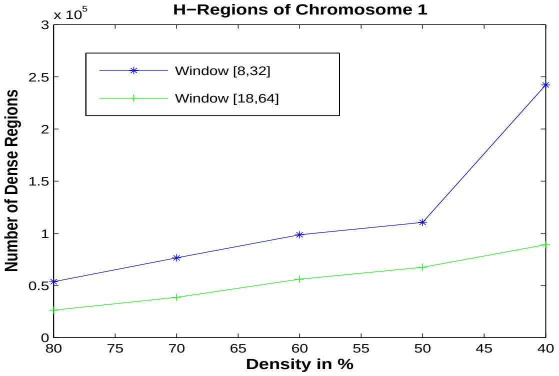


Figure 6. Number of H-regions of Chromosome 1 with Respect to Different Density Values.

5.3 Extracting Frequent Arrangements

Finally, an efficient mining algorithm, as described in Section 4, has been applied on the extracted H-regions to detect frequent arrangements between them. Specifically, the algorithm has been applied on the extracted H-regions of Chromosomes 1, 2 and 38 of the Canis Familiaris. The size of the sliding window was 1000. We managed to extract an interesting number of frequent patterns. In all three cases a great number of overlaps and contains has been detected between H-regions of C and G. These regions are in fact the $G + C$-rich ones, and the CpG islands.

Another observation was that in both Chromosomes 1 and 2 there was a high frequency ($\approx 65\%$) of overlaps between bases A and C. In Chromosome 38 we detected a high frequency of: (1) follows of (A, G) and T ($\approx 72\%$), (2) overlaps of C and G, with follows of G and T, and follows of C and T ($\approx 52\%$), (3) meets of C and T, with follows of G and T, and follows of C and T ($\approx 56\%$). Figure 5 gives a sample of the frequent arrangements that have been extracted for Chromosomes 1, 2 and 38 of the *Canis Familiaris*.

By and large, there are a great number of arrangements detected for these Chromosomes, with the minimum support threshold varying from 60% to 40%. Chromosome 38 gave the greatest number of arrangements despite the fact it was the smallest (in length) chromosome examined. The size of the extracted arrangements was limited between two and three. Note that in the above experiments the size of H-regions was in the range $[18, 64]$. When the algorithm was applied on the smaller H-regions (of size $[8, 32]$) it was noticed that: (1) the number of arrangements increased, (2) there was a significant increase in the number of relations of type $follow$. The latest is expected, since the smaller the size of event intervals, the greater the chance of a follow to occur.

6 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 two efficient algorithms to solve it. The first algorithm applies an efficient segmentation technique that splits the original sequence to a set of segments. The key idea of the second 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. After identifying these regions, the sequence is converted to a sequence of labeled intervals (each one corresponding to a region). We further applied an efficient arrangement mining algorithm to extract the complete set of frequent arrangements of the extracted regions found in an experimental evaluation of our algorithms on the dog genome.

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

References

[1] J.F. Allen and G. Ferguson. Actions and events in interval temporal logic. Technical Report 521, The University of Rochester, July 1994.

[2] I. E. Auger and C. E. Lawrence. Algorithms for the optimal identification of segment neighborhoods. *Bulletin of Mathematical Biology*, 51:39–54, 1989.

[3] T. R. Bement and M. S. Waterman. Locating maximum variance segments in sequential data. *Mathematical Geology*, 9:55–61, 1977.

[4] P. Bernaola-Galvan, R. Roman-Roldan, and J. L. Oliver. Compositional segmentation and long-range fractal correlations in dna sequences. *Physical Review E*, 53:5181–5189, 1996.

[5] J. V. Braun, R. K. Braun, and H. G. Mueller. Multiple change-point fitting via quasi-likelihood, with application to dna sequence segmentation. *Biometrica*, 87:301–314, 2000.

[6] J. V. Braun and H. G. Mueller. Statistical methods for dna segmentation. *Statistical Science*, 13:142–162, 1998.

[7] E. Carlstein, H. G. Mueller, and D. Siegmund. Change-point problems. *Lecture Notes and Monograph Series*, 23(2), 1994.

[8] G. A. Churchill. Stochastic models for heterogeneous dna sequences. *Bulletin of Mathematical Biology*, 51(1):79–94, 1989.

[9] G. A. Churchill. Hidden markov chains and the analysis of genome structure. *Computes and Chemistry*, 16(2):107–115, 1992.

[10] J. W. Ficket, D. C. Torney, and D. R. Wolf. Base compositional structure of genomes. *Genomics*, 13:1056–1064, 1992.

[11] C. Freksa. Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1):199–227, 1992.

[12] Y-X. Fu and R. N. Curnow. Maximum likelihood estimation of multiple change points. *Biometrica*, 77:563–573, 1990.

[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*, 65:041905, 2002.

[14] F. Larsen, G. Gundersen, R. Lopez, and H. Prydz. Cpg islands as gene markers in the human genome. *Genomics*, 13:1095–1107, 1992.

[15] W. Li, P. Bernaola-Galvan, H. Fatameh, and I. Grosse. Applications of recursive segmentation to the analysis of dna sequences. *Computes and Chemistry*, 26(2):491–510, 2002.

[16] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurences. In *Proc. of ACM SIGKDD*, pages 146–151, 1996.

[17] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos. Discovering frequent arrangements of temporal intervals. In *Proc. of IEEE ICDM*, pages 354–361, 2005.

[18] J. C. Venter. The sequence of the human genome. *Science*, 291:1304–1351, 2001.

[19] C-T. Zhang, F. Gao, and R. Zhang. Segmentation algorithm for dna sequences. *Physical Review E*, 72:041917, 2005.