Adapting Machine Learning Technique for Periodicity Detection in Nucleosomal Locations in Sequences

Faraz Rasheed[1], Mohammed Alshalalfa[1], Reda Alhajj[1,2]

[1]Computer Science Dept, University of Calgary, Calgary, Alberta, Canada
[2]Department of Computer Science, Global University, Beirut, Lebanon

Abstract. DNA sequence is an important determinant of the positioning, stability, and activity of nucleosome, yet the molecular basis of these remains elusive. Positioned nucleosomes are believed to play an important role in transcriptional regulation and for the organization of chromatin in cell nuclei. After completing the genome project of many organisms, sequence mining received considerable and increasing attention. Many works devoted a lot of effort to detect the periodicity in DNA sequences, namely, the DNA segments that wrap the Histone protein. In this paper, we describe and apply a dynamic periodicity detection algorithm to discover periodicity in DNA sequences. Our algorithm is based on suffix tree as the underlying data structure. The proposed approach considers the periodicity of alternative substrings, in addition to considering dynamic window to detect the periodicity of certain instances of substrings. We demonstrate the applicability and effectiveness of the proposed approach by reporting test results on three data sets.

1 Introduction

Eukaryotic DNA is complex with basic histone proteins forming nucleosome and higher order chromatin. The ability of the histone octamer to wrap differing DNA sequences into nucleosomes is highly dependent on the specific DNA sequence. Those sequence fragments that wrap Histone proteins got a lot of attention because it is of great importance to determine the positioning of nucleosome which affects gene regulation positively and negatively.

In this work we consider DNA sequence as a time series data. Time series is a sequence of data values collected usually at a uniform time interval. Real-life applications have several examples like number of transactions per hour in a superstore, hourly weather data for particular location, etc. We detect the periodicity of certain sequences (dinucleotides) within a given DNA sequence. Periodicity mining in time series data and in DNA sequence in particular is a hot area of research. Our approach employs suffix trees as the basic data structure; we map the DNA sequence into suffix tree based representation, which is then used to find the periodicity in the DNA sequence.

Given the sequence of many DNA segments, the problem is to find the periodicity of certain sequences along the whole DNA sequences. Periodicity detection in biological data is different in many ways from the traditional time series data. First,

there is the concept of alternatives where a group of strings can replace each other. The task is to find the periodicity of a group by considering alternative strings. For instance, in our experiments the strings TA, TT and AA are parts of an alternative group, and the presence of any of these is counted as valid repetition. Hence, the sequence TTACGAATGGTAGT has the periodicity for alternative string group (AA, TT, TA) with period = 5 starting at position 0 and periodic strength = 1 (or 100%).

Secondly, we introduce the concept of a window for detecting periodic occurrences. In traditional time series data, we expect to find the periodic string (or symbol) at positions *stPos*, *stPos*+*p*, *stPos*+*2p*, and so on, where *stPos* is the starting position of the periodic pattern while *p* is period value. For example, in the sequence abdadbacc, 'a' is periodic with period $p = 3$ starting from $stPos = 0$ with periodic strength of 100%. Hence, we expect to find 'a' to appear at index positions 0, 3, 6, …, which are *stPos*, *stPos*+*p*, *stPos*+*2p*, …. Unfortunately, DNA sequences generally do not repeat that strictly. For example, in the chicken nucleosomal sequence that we used in our experiments, the alternative string group (AA, TT, TA) is expected to be repeated with the periodicity of 10, but it is not guaranteed that the alternative string group would repeat after every 10 index positions; rather the alternative string group may appear ±2 positions away from the expected periodic difference of 10. This means that if the first appearance of the alternative group (say TT) is found at position 6 and the expected period is 10, then the second appearance of the alternative group (say TA) may be found from position 14 to position 18 (16±2), if the relaxed occurrence range (or window) is taken as 2. Formally, the appearance of the occurrence string is considered as valid if it is found at index position *stPos*, $stPos + p \pm RR$, $stPos + 2p \pm RR$, $stPos \pm 3p + RR$, …, where *RR* is the relaxed range window under which the occurrence would be considered valid.

Thirdly, when applying periodicity detection algorithm on DNA sequence, we are only interested in the periodicity of a specific string (alternative string group) and not in the periodicity detection of all substrings and symbols. Further, we are only interested in a specific period range and not in all the periods.

While achieving the third requirement might be possible with some of the periodicity detection algorithms with few adjustments, most of them, especially the ones which work on the shift & compare principle, e.g., [1, 2], cannot meet the first two requirements. Our algorithm is flexible enough to fulfill these requirements as it uses suffix tree and keeps the occurrence vector for each repeating sub-sequence.

Suffix tree based data structure allows us to quickly find all the occurrences of all repeating sub-sequences (sub-strings) in linear time. Since we have the separate occurrence vector for each sub-sequence, we may combine the occurrences of each string in the alternative string group into one vector. As our algorithm calculates the periodicity using the occurrence vector of a sub-string, we can run the algorithm for periodicity calculation on the combined occurrence vector of alternative string group. Since we validate the repetition by taking the mod of occurrence position with starting position (as can be seen in the algorithm presented in Figure 3), we can also accommodate the relaxed range occurrence concept by simply checking if a particular occurrence falls in the relaxed range window of the expected periodic position.

The rest of the paper is organized as follows. Related work is presented in Section 2. Section 3 contains the description of the approach. Results and discussion is presented in Section 4. Section 5 is conclusions.

2 Related Work

Periodicity detection both in time series databases, e.g., [2] and in DNA sequences, e.g., [18] is getting more active and more significant. As finding periodicity in time series databases in concerned, Indyk *et al* [1] presented their periodic trends algorithm which can find the segment periodicity only, i.e., to detect if the entire time series can be achieved by the repetition of a sequence of symbols. Elfeky *et al* [2] presented two algorithms to find symbol and segment periodicities in the time series, but their algorithm favors shorter periods. On the other hand, our algorithm is capable of detecting symbol, segment, and sequence periodicity. The algorithm does not favor any period size. Unlike the work of Elfeky, we do not need two separate algorithms to find the symbole and segment periodicities. Our single algorithm can detect segments, symbol and sequence periodicities by just a single pass over the data once the series is represented in suffix tree.

To the best of our knowledge, no existing algorithm of periodicity detection can fulfill all the three requirements outlined in the previous section. We elaborate on this by comparing our algorithm with the three well-known algorithms, namely, Indyk [1], Elfeky [2,14], and Ma [15] for the adaptablity to work with biological data.

For the first requirement, the concept of alternatives where a group of strings can replace each other, Indyk [1] and Elfeky [14] can not be applied as they find the segment periodicity of entire time series and not the for a single or a sequence of alphabets. Elfeky [2] and Ma [15] can find the symbol periodicity, which is the periodicity of a single symbol, but can not find the periodicity of a sequence of symbols. So, if we wish to use this algorithm to find the periodicity of 'TA', we need to calculate the periodicity of 'T' and 'A' separately, and then combine the periodicity results using Han's approach [16]; still it won't give the exact periodic strength of 'TA', rather it would give the (min-max) range for the periodic strength. If we wish to implement the alternative strings concept in [2], we need to replace all occurrences of each string in alternative strings group by a single string and then we can run the algorithm to find the periodicity of that string. Clearly, there is disadvantage in this approach as we have to make a copy of the entire time series for each alternative string group.

For the second requirement, the concept of time tolerance (or relaxed range) window for periodic occurrences, it is used to accommodate various types of noise (insertion, deletion, replacement or a mixture of these) in the data. Since Indyk [1] does not calculate the symbol (or sequence) periodicity, it can not be used in this situation. Elfeky [2] algorithm can not accommodate the time tolerance window concept as it shifts the entire time series for each candidate period. Recently, Elfeky *et al* presented a different algorithm called 'WARP' [14] to deal with this problem. But their new algorithm can only find segment periodicity (periodicity for the entire time series) and not symbol periodicity. Even if we somehow generate the DTW matrix for each alternative string, it would worsen the complexity of the algorithm (which is already $O(n^2)$ compared to their convolution algorithm's [2] complexity $O(n \log n)$). The algorithm of Ma *et al* [15] can find the symbol periodicity with time tolerance window, but can not find the sequence periodicity and would have to combine the individual periodicity results for symbols to generate the approximate periodicity of the sequence where the periodic strength is given in (min-max) range. Since we run

the algorithm on the occurrence of the entire string (like 'TA' or 'AA'), we provide the exact periodic strength for the strength and not in some min-max range.

The third requirement, i.e., calculating the periodicity for a specific set of alternative strings, can be done by Elfeky's [2] and Ma's [15] algorithms, but we have to combine the results of the periodicity of individual symbols and join them by considering their starting positions in the time series; this would give us the periodic strength in (min-max) range. Hence, we may conclude that our algorithm is more flexible than any other periodicity detection approach to be applied to the biological data. The algorithm can be easily modified to work with the DNA sequences in addition to working for the regular time series data.

The periodicity detection in DNA sequence is of great significant in order to detect the nucleosome positioning. Many works have shown that there is dinucleotides periodicity in nucleosome–positioning sequences. Bina [3] demonstrated that there is periodic signal for AA/TT (10.26 bp) and GG/CC(10.bp) and AA dominates the occurrence of AA/TT. These results were obtained from the analysis of nucleosome sequences derived from simian virus 40 chromatin. Bina used statistical approach, which was based on the alignment of the DNA fragments with respect to their midpoints. Satchwell et al [4] detected a periodicity of 10bp in chicken nucleosome . They used fourier transformation to analyse the sequences. In another work, Herzel *et al* [5] reported that correlation functions of complete genomes revealed pronounced oscillations with period in the range of 10-11bp. Thastrom *et al* [6] discover periodicity in DNA sequences using ClUSTAL W for multiple sequence alignment. Segal *et al* [7] used probabilistic nucleosome-DNA model within statistical framework to compute the nucleosome organization intrinsic to the genomic DNA sequence. Interestingly, dinucleotide periodicity is detected in prokaryotes sequences, although they do not have nucleosomes [8]. Hosid [9] showed that sequence periodicity close to 11 is detected in *E.coli* . Only AA/TT dinucleotides contributed to overall dunucleotide periodicity in intergenic regions.

3 Periodicity Detection

Motivated by the above analysis, we developed a novel approach for periodicity detection by integrating suffix trees in the process. The proposed approach involves several phases as described next.

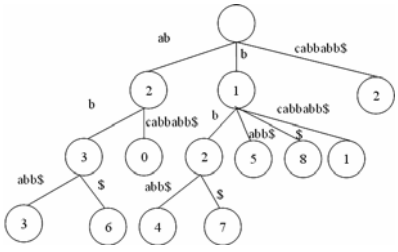

Fig. 1. The suffix tree for the string abcabbabb$.

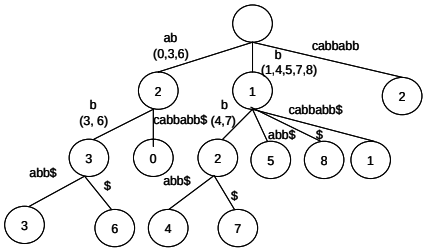

Fig. 2. The suffix tree for the string abcabbabb$ with substring occurrences.

First Phase: Suffix Tree Based Representation: Suffix tree is a famous data structure [10] that has been proved to be very useful in string processing, e.g., [10, 11]. It can be efficiently used to find a substring in the original string, find the frequent substring; it can also be used to solve other substring matching problems. Each of the branches of the suffix tree represents a suffix of the original string. Hence, a suffix tree for a string of length 'n' has 'n' branches, and thus 'n' leaf nodes. For example, Figure 1 shows the suffix tree for the string 'abc abb abb$' where $ denotes the terminating symbol.

Each leaf node in the tree has an integer value showing the starting position of the substring achieved through the path from root to that leaf in the original string. Since there are exactly 'n' suffixes for a string, each starting at one of the index positions, there are 'n' leaf nodes in the tree. Each internal node (nodes that are neither leaf nor the root) has the integer value representing the length of the substring so far achieved while traversing from the root to the node. A suffix tree can have a maximum of 2n nodes, but mostly having periodicity and repetition in the time series, there are less than '2n' nodes in the suffix tree for these series.

We use the famous Ukonen algorithm [17] to construct the suffix tree for a given time series, which runs in linear time. The algorithm gives us the collection of 'edges', each having the starting node number, end node number, the first character index and the last character index and the value. For example, the edge from the root with label 'ab' in Figure 1 is represented as: starting node number: 0, end node number: 1, first character index: 0, last character index 1, and the value: 2. Thus, this edge can be represented in five-tuple (0, 1, 0, 1, 2). The subsequent edge labeling b is represented as (1, 4, 2, 2, 3), and the edge labeling cabbabb$ is represented as (1, 5, 2, 9, 0).

Second Phase: Periodicity Detection: Once we have the decorated suffix tree, we invoke the periodicity detection algorithm given in Figure 3. The performed process traverses the tree in bottom up fashion. During the traversal, each leaf node passes its value to the parent. The internal nodes after receiving the values from all of their children collect these in the collection called the *occurrence vector*. An occurrence vector is represented in our algorithm as 'occur_vect'. The tree in Figure 1 after performing this step is presented in Figure 2, where each internal node has its own occurrence vector. In fact, this vector shows the index positions in the original time series where this sequence appear. Since there are a maximum of 2n nodes in the suffix tree of a string of length 'n' and there are 'n' leaf nodes in the suffix trees, the tree should have significantly less than 'n' such vectors.

The second step is to calculate another vector for each of these occurrence vectors, which we call the difference vector (or 'diff_vect' in our algorithm). Let V be the occurrence vector of length m; $V = v_0, v_1, \ldots, v_{m-1}$

The difference vector 'D' would always have the length m-1 and would be

$$D = v1\text{-}v0, v2\text{-}v1, \ldots, vm\text{-}1\text{-}vm\text{-}2$$

Table 1. Occurrence and difference vectors for the sequence 'ab'.

Index	0	1	2	3	4	5	6	7	8	9
occur_vect	0	3	12	16	21	24	27	38	45	48
diff_vect	3	9	4	5	3	3	11	7	3	

This is calculated by simply taking the difference of the consecutive values, and thus called the difference vector. The difference vectors contain the candidate periods. Each of these periods (with some exceptions mentioned in the sequel) is checked and the corresponding periodic strength is calculated.

Let vj and sj represent the jth entry in the difference and occurrence vectors, respectively, and 'i' be a positive integer. Then, we increment count(p, st) by 1 if and only if sk = sj + ivj, where i×vj ≤ max(occur_vect). The count(p, st) represents the frequency of the occurrence of a sequence starting from 'st' with a period value 'p'.

If the length of the time series is n, then the periodicity strength 'τ' is calculated as:

$$\tau(p, st) = \frac{count(p, st)}{\left\lfloor \frac{n\text{-}st}{p} + y \right\rfloor} \qquad y = \begin{cases} 1 & \text{iff } ((n\ st) \bmod p) > \text{edges value} \\ 0 & \text{otherwise} \end{cases}$$

The periodicity strength is the ratio between the frequency of a sequence's occurrences and the maximum possible number of occurrences for that sequence. For example, for the sequence abcabbabc$, τ(3, 2, 'c') is 2/3, as there are 2 occurrences of 'c', while the maximum possible occurrences are 3.

Adjustments to make the algorithm work with DNA Sequences: The already described algorithm [13] is not ready to deal with the DNA sequence data because of the special properties of the DNA sequences discussed earlier in Section 1. But the algorithm is flexible enough to work with such data with some minor changes. We are not looking for one sequence (or substring say TT) to be repeated at certain positions; however, we look for the occurrence of any of the alternative strings (AA/TT/TA). This is achieved by keeping a separate collection for alternative string occurrences and by applying the periodicity detection algorithm only on this combined collection.

Secondly, we adapted the window range of the periodic appearance of the string when analyzing the occurrence vector (as can be seen in line 2.9.1 of the CalculatePeriod algorithm in Figure 3). Hence, if the detected period is 10 then we also count the occurrence at a position which is in the range of 8-12bp from the occurrence of the pervious sequence (AA/TT/TA). Another problem is that not every occurrence of AA/TT/TA is to be periodic. Some of these instances is just randomly embedded in between two periodic instances. Our algorithm is able to deal with such problems. We deal with noise problem as follows, we measure the distance between the occurrence of any of AA/TT/TA instances and the following instance, if it is less than 8, then we ignore the second instance temporarily and check the distance with the instance which follows till we reach an instance in the range of 8-12 (as can be seen in line 2.3 of the CalculatePeriod algorithm in Figure 3).

The algorithm also does not consider periods greater than a certain value. It calculates the periodicity starting from each and every instance of the altenative string. This way, we do not miss any instance to be considered in the periodicity. Since we are looking for periodicity over the complete fragment, we ignore the period starting after a specific index position. The algorithm can also detect the periodicity strength, which allows us to include only the stonger periods (say those having periodic strength over 0.6).

To demonstrate the functionality of the algorithm, let us consider a nucleosomal DNA sequence of chicken as provided in [7]. This sequence contains 145 nucleotides (or it is 145 characters long). The sequence along with the periods found by the

algorithm is presented in Figure 4. All the occurrences marked by either +, * or – are added to the combined occurrence collection which is like

```
1. Initialize rootOccurSt and rootOccurLength and stack ‘s’
2. With each children edge ‘e’ (having stn = 0) of the root edge
   2.1. Sort children edges
   2.2. e.pntVal = 0              // parent value
   2.3. e.pntOccurSt = rootOccurSt
   2.4. e.pnOccurLength = rootOccurLength
   2.5. push e to stack ‘s’
3. while (stack is not empty)
   3.1. e = s.pop()
   3.2. if edge is already marked
      3.2.1. ProcessEdge(e)
      3.2.2. if e.pntOccurSt is blank
          3.2.2.1. e.pntOccurSt = e.occurSt
          3.2.2.2. e.pntOccurLength = e.occurLength
      3.2.3. else Join&Sort(e.pntOccurSt, e.pntOccurLength,e.occurSt, e.occurLength)
   3.3. else if edge has not been marked yet
      3.3.1. if e leads to leaf e.val = N-(e.lci-e.fci) + 1 + e.pntVal
          3.3.1.1 occur.add(e.val)
      3.3.2. else e.val = e.lci - e.fci + 1 + e.pntVal
          3.3.2.1. find and sort all children edges of e
          3.3.2.2. With each child edge ‘ce’
             ce.pntVal = e.val
             ce.pntOccurSt = e.occurSt
             ce.pnOccurLength = e.occurLength
             s.push(ce)
      3.3.3. mark ‘e’
4. Initialize an Edge ebio and set ebio.occurSt = bioOccurList,
      ebio.occurLength = bioOccurList.length,
      ebio.value = length of any alternative string
5. CalculatePeriod(ebio)
ProcessEdge: Edge e
1. Initialize chkStr = T.subString(e.occurSt, e.val)
2. if(chkStr matches any of alternatives)
   2.1. Add all e.occurLength number of occurences starting from
        e.occurSt to bioOccurList
CalculatePeriod: Edge e
1. current = e.occurSt
2. for ( i = 1; i < e.occurLength; i++)
   2.1. diffVal = current.next.val - current.val
   2.2. Initialize bioCurr = current.next
   2.3. while(diffValue < minPeriodValue AND bioCurr != null)
      2.3.1. diffValue = bioCurr.Value - current.value
      2.3.2. bioCurr = bioCurr.next
   2.4. if (diffVal < e.val OR diffVal > maxPeriodVal OR current.val > minStPos)
      2.4.1 current = current.next; continue from 2
   2.5. initialize p as candidate period
   2.6. p.val = diffVal, p.stPos = current.val, p.fci = p.stPos, p.len = e.val
   2.7. modRes = p.stPos mod p.val
   2.8. subCurrent = current, preSubCurValue = -5
   2.9. for ( int j = i; j<=e.occurLength; j++)
      2.9.1. if(modRes >= ((subCurrent.value mod p.periodValue) - relaxedRange)
           AND modRes <= ((subCurrent.value mod p.periodValue) + relaxedRange))
          2.9.1.1. if( (subCurrent.value - preSubCurValue) > 2*relaxedRange-1)
             2.9.1.1.1. p.freq++
             2.9.1.1.2. preSubCurValue = subCurrent.value
      2.9.2. subCurrent = subCurrent.next
   2.9. if ( (T.Len - 1 - p.stPos) mod p.val >= e.val) y = 1 else y = 0
   2.10. p.th = p.freq / Floor( (T.Len - 1 - p.stPos) / p.val + y)
   2.11. if (p.th >= minThreshold) add p to PeriodCollection
   2.12. current = current.next
```

Figure 3. Algorithm for Periodicity Detection

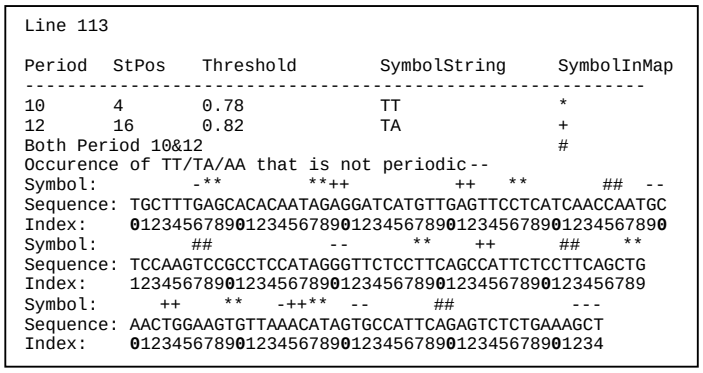

Figure 4 The chicken sequence and corresponding periods

Occur_vect(TT/AA/TA) = (3, 4, 14, 16, 28, 33, 42, 46, 54,…, 139, 140).

Now the difference between any two occurrence is a candidate period. The first period found by the algorithm is 10, starting at position 4. Its valid occurrences are

T(p, stPos) = T(10, 4) = 4, 14, 33, 42, 54, 72, 86, 92, 106, 114, 126

$\tau(10,4) = 11 / \lfloor ((144 - 4) / 10) \rfloor = 11/14 = 0.78$

Note that positions 33 and 42 are considered as valid because they occur inside ± 2 window of the exact positions 34 and 44, respectively.

4 Experimental Evaluation

For the experimental evaluation of the algorithm, we have used chicken and yeast data sets from [7]. These data sets contain 177 and 199 fragments of DNA, respectively, each is a nucleosome positioning sequence of around 150bp. In order to check the correctness of our algorithm, we used another dataset which contains random sequences from different chromosomes of chicken. We are concerned about the periodicity of AA/TT/TA in a dynamic window of 8 to 12 nucleotides.

The chicken and yeast data are known to have the periodicity of ~10 for AA/TT/TA. After applying the algorithm to the 177 chicken sequences, we got all possible periodicities for AA/TT/TA within each fragment. Then, we calculated the average of the periodicities for each fragment and the average for the whole set of fragments. For both chicken and yeast datasets, we were able to find the periodicity of AA/TT/TA in about 90% of the sequences, and the average periodicity is ~9.4 for chicken and ~9.3 for yeast. We also could apply the algorithm to the random dataset and only about 10% of the sequences showed periodicity. We believe this small percentage occurred by chance because some of the random sequences might be selected with nucleosome–positioning sequences. The percentage of each of the datasets showing the periodicity is depicted in Figure 5.

Many studies have shown that the periodicity of AA/TT in the eukaryotic is ~ 10. This periodicity has a lot of biological meaning in addition to its meaning in time series analysis. In a “relaxed” double-helical segment of DNA, the two strands twist

around the helical axis once every 10.55bp of the sequence. If a DNA segment under twist strain were to be closed into a circle by joining its two ends and then it is allowed to move freely, the circular DNA would contort into new shape, such as a simple figure-eight. Such a contortion is a supercoil. DNA supercoiling is important for DNA packaging within all cells. Because the length of DNA can be thousands of times that of a cell, packaging this genetic material into the cell or nucleus (in eukaryotes) is a difficult feat. Supercoiling of DNA reduces the space and allows for a lot more DNA to be packaged. Extra helical twists are positive and lead to positive supercoiling, while subtractive twisting causes negative supercoiling. According to Crick's formula for helicalDNA trajectories, periods above 10.55bp generate negatively supercoiled DNA, whereas lower periods induce positive supercoiling. Therefore, sequence periodicities reflect the characteristic superhelical density of genome DNA.

Our results confirm that periodicity of AA/TT/TA is around 10bp, which agrees with all the previous results reported by statistical approaches [3, 4, 7]. Positive supercoiling has shown to inhibit gene expression [12]. This result indicates that the nucleosomal DNA is not an active region. Since the hot regions which are transcribed continuously will be most of the time free from Histone proteins to let other proteins like RNA polymerase to access the DNA code. However, when the DNA is positively supercoiled, that means it will stay bounded to the histone proteins which means no transcription process will take place. We think that the periodicity of housekeeping genes for example will be greater than 10.55bp since it is active all the time.

We also argue that the periodicities detected in the chicken and yeast are not by chance because the algorithm did not discover any periodicity in the tested random dataset as expected. This means that our algorithm was able to deal with DNA sequences to identify periodicities when they do exist.

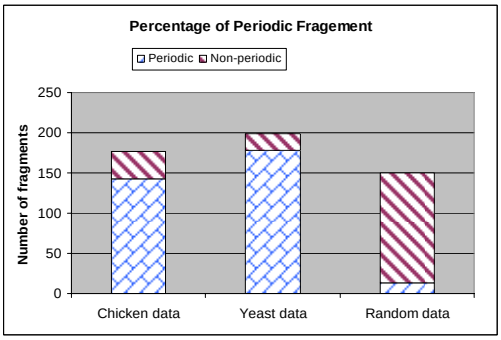

Figure 5. Percentage of Periodic Fragment

5 Conclusions

DNA sequence analysis is getting more popular among biologists and statisticians. Most of the approaches to analyze DNA sequence are based on statistical techniques. In this work, we have applied an algorithm which is used in time series analysis to

detect periodicity in DNA sequence. We considered DNA sequence as time series data, and the occurrence of each nucleotide is an event at certain time. This way, we demonstrate that our approach is flexible enough to deal with DNA sequence which is more challenging than time series data. Currently, we are extending the algorithm to automatically work with both time series and DNA sequences. Also, we are planning to apply the algorithm on whole genome sequences.

References

1. P. Indyk, N. Koudas, and S. Muthukrishan, "Identifying representative trends in massive time series data sets using sketches", *Proc. of VLDB*, 2000.
2. M.G. Elfeky, W.G. Aref, and A.K. Elmagramid, "Periodicity detection in time series databsaes," *IEEE TKDE*, Vol.17, No.7, pp.875-887, 2005.
3. M. Bina, "Periodicity of dinucleotide in Nucleosomes derived from simian virus 40 chromatin, *Journal of Molecular Biology*, 235, pp.198-208, 1994.
4. S.C. Sarchwell, et al, "Sequence periodicities in checkin nucleosome core DNA'', *Journal of Molecular Biology*, 191, pp.659-675, 1986.
5. H. Herzel, O. Weiss, and N. Trifonov, "10-11 bp periodicities in complete genomes reflect protein structure and DNA folding", *Bioinformatics*, vol.15, no.3, pp.187-193, 1999.
6. A. Thastrom, L.M. Bingham, and J. Widom, "Nucleosomal locations of dominant DNA sequence motifs for histone-DNA interaction and nucleosome positioning," *Journal of Molecular Biology*, 338, pp.695-709, 2004.
7. E. Segal, et al, "Agenomic code for nucleosome positioning", *Nature*, vol.442, no.17, pp.772-778.
8. H. Herzel, O. Weiss, and E.N. Trifonov, "Periodicity in complete genome of archaea suggests positive supercoiling", *Journal of Biomol Struct Dyn*, 16, pp.341-345, 1998.
9. S. Hosid, et al, "Sequence periodicity of Escherichia coli is concentrated in intergenic regions," *BMC Molecular Biology*, vol.5, no.14, 2004.
10. D. Gusfield. Algorithms on Strings, Trees, and Sequences. Cambridge Univ. Press, 1997.
11. R. Grossi and G. F. Italiano, "Suffix trees and their applications in string algorithms", *Proc. of South American Workshop on String Processing*, pp.57-76, 1993.
12. M.R. Gartenberg and J.C. Wang, "Positive supercoiling of DNA greatly diminishes mRNA synthesis in yeast," *PNAS*, vol.89, no.23, pp.11461-11465, 1992.
13. F. Rasheed, R. Alhajj, "Using suffix trees for the periodicity detection in time series databases", Technical Report, Dept of Computer Science, University of Calgary, May 2007.
14. M.G. Elfeky, W.G. Aref, and A.K. Elmagarmid, "WARP: Time Warping for Periodicity Detection," *Proc. of IEEE ICDM*, pp.138-145, 2005
15. S. Ma, J. L. Hellerstein, "Mining partially periodic event patterns with unknown periods," *Proc. of IEEE ICDE*, pp.205-214, 2001.
16. J. Han, Y. Yin, and G. Dong, "Efficient Mining of Partial Periodic Patterns in Time Series Database," *Proc. of IEEE ICDE*, p.106, 1999.
17. E. Ukkonen. "Online Construction of Suffix Trees", *Algorithmica*, 14(3):249-260, 1995.
18. M. Ahdesmäki, H. Lähdesmäki and O.,Yli-Harja, "Robust Fisher's test for periodicity detection in noisy biological time series," *Proc. of IEEE International Workshop on Genomic Signal Processing and Statistics,* Tuusula, FINLAND, 2007.