TELKOMNIKA, Vol.17, No.3, June 2019, pp.1352~1359
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v17i3.11750

Parallel random projection using R high performance computing for planted motif search

Lala Septem Riza*[1], Tyas Farrah Dhiba[2], Wawan Setiawan[3], Topik Hidayat[4], Mahmoud Fahsi[5]
[1,2,3]Department of Computer Science Education, Universitas Pendidikan Indonesia, Indonesia
[4]Department of Biology Education, Universitas Pendidikan Indonesia, Indonesia
[5]EEDIS Laboratory, Djillali Liabes University, BP 89 22000 Sidi Bel Abbes, Algeria
*Corresponding author, e-mail: lala.s.riza@upi.edu

Abstract

Motif discovery in DNA sequences is one of the most important issues in bioinformatics. Thus, algorithms for dealing with the problem accurately and quickly have always been the goal of research in bioinformatics. Therefore, this study is intended to modify the random projection algorithm to be implemented on R high performance computing (i.e., the R package pbdMPI). Some steps are needed to achieve this objective, ie preprocessing data, splitting data according to number of batches, modifying and implementing random projection in the pbdMPI package, and then aggregating the results. To validate the proposed approach, some experiments have been conducted. Several benchmarking data were used in this study by sensitivity analysis on number of cores and batches. Experimental results show that computational cost can be reduced, which is that the computation cost of 6 cores is faster around 34 times compared with the standalone mode. Thus, the proposed approach can be used for motif discovery effectively and efficiently.

Keywords: *bioinformatics, high performance computing, motif discovery, planted motif search, R programming language*

1. Introduction

Motif discovery in DeoxyriboNucleic Acid (DNA) sequences is one of the most important issues in the field of bioinformatics since it may help biologists to obtain better understanding on the structure and function of the molecules in the sequence [1]. A motif is a short pattern that repeats in the DNA sequence consisting of a combination of four basic nitrogen: Adenine (A); Guanine (G); Cytosine (C); and Thymine (T) [2]. Issues in motif discovery can be categorized into 3 types, namely Simple Motif Search (SMS), Edit distance based (EMS), and Planted Motif Search (PMS) [3]. The purpose of SMS is to find all the motifs from lengths 1 to the specified length in all sequences of [4] while the purpose of the EMS is to find all the motifs on the desired number of sequences [5]. PMS aims to find the motive that appears in every sequence that exists [6].

In PMS, there are two important input parameters: the desired length of motif symbolized by$l$and the number of mismatches denoted by d [7]. For example, there are three DNA sequences, as follows: S_1=ATTGCTGA, S_2=GCATTGAA, and S_3=CATGCTTG. With$l$=4 and d=1, we obtain the following repetitive motifs: ATTG and TTGC. It can be seen that PMS is included in the NP-Hard problem, so that if this algorithm is run to look for all possible motives that appear in all sequences, then the time spent will be exponential [1]. Random Projection (RP) [8] is one of the algorithms used for motive search problems in DNA sequences included in PMS. In this algorithm a piece of the input data in the form of sub-sequences (l-mers) will be projected according to the random position determined by k (k-mers) values [1, 9]. RP represents that mutations can occur anywhere so the projection is done randomly. Even though many algorithms have been introduced, since PMS is NP-hard problems, an implementation of the algorithms into parallel computing is necessary to be done.

Therefore, this research is aimed to design and implement RP for dealing with PMS in parallel computing in R programming language. The R programming language [10] is chosen since it has become the de-facto standard for statistics, data analysis, and visualization.

Received November 10, 2018; Revised January 29, 2019; Accepted February 28, 2019

Nowadays, there are many algorithms, collected in software libraries/packages, that have been implemented and saved in the Comprehensive R Archive Network (CRAN) at https://cran.r-project.org/. In this repository, one of packages in R used for high performance computing and big data analysis is pbdMPI [11] that is used in this research.

In the literature, we found some relevant articles discussing implementations of motif discovery in parallel computing. For example, in Clemente & Adorna's study [12], random Projection algorithm was developed in the concept of GPU (Graphic Processing Units). Each processor will be directed into threads that work within the device or GPU. Meanwhile, the sequential process will be executed on the host or CPU. TEIRESIAS has been introduced to improve the speed on finding maximal pattern [13]. An enhancement of the PMSPRUNE algorithm has been proposed with two additional features: neighbor generation on a demand basis and omitting the duplicate neighbor checking [14]. Furthermore, there are some different approaches for dealing with patterns matching in various fields. For instance, multiple patterns matching methods was introduced for large multi-pattern matching [15]. Improving the scanning mode of Square Non-symmetry and Antipacking Model (SNAM) for binary-image is obtained by proposing the new neighbor-finding algorithm [16].

The rest of the paper is organized as follows: first, the global procedure of this research is presented in section 2. In section 3, a main contribution, which is a modification and implementation of parallel random projection by using the pbdMPI package, is discussed. To validate and analyze the proposel computational model, we conduct some experiments in section 4 and some analysis in section 5. Finally, we conclude the research in section 6.

2. Research Method

Figure 1 shows the research design done in this study. It can be seen that first, we perform some preparation, such as identifying problems, research objectives, and literature study. These activities have been presented in the previous section. Then, we present a main contribution of this research, which is designing and implementing parallel random projection with R high performance computing (i.e., the pbdMPI package). This part will be explained in the next section. After that, we conduct some experiments and their analysis of the results. Drawing some conclusión is presented in the end.

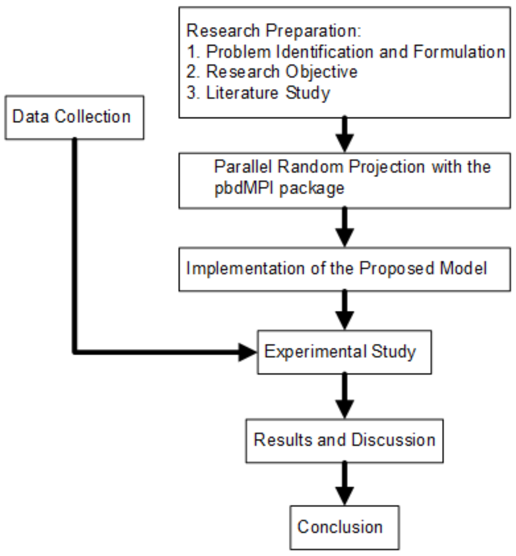

Research Preparation:
1. Problem Identification and Formulation
2. Research Objective
3. Literature Study
Data Collection
Parallel Random Projection with the pbdMPI package
Implementation of the Proposed Model
Experimental Study
Results and Discussion
Conclusion

Figure 1. Research design to conduct parallel random projection

3. Parallel Random Projection with the pbdMPI package

Basically, the computational model proposed in this research can be seen in Figure 2. First, after reading and converting the input data from the .falsa file, we perform a modification of random projection by utilizing R high perfomance computing (i.e., the pbdMPI package), called parallel random projection with pbdMPI. Detailed explanation regarding the proposed approach can be seen in Figure 3. The results of this model is all motifs, their starting indices, and computational costs.

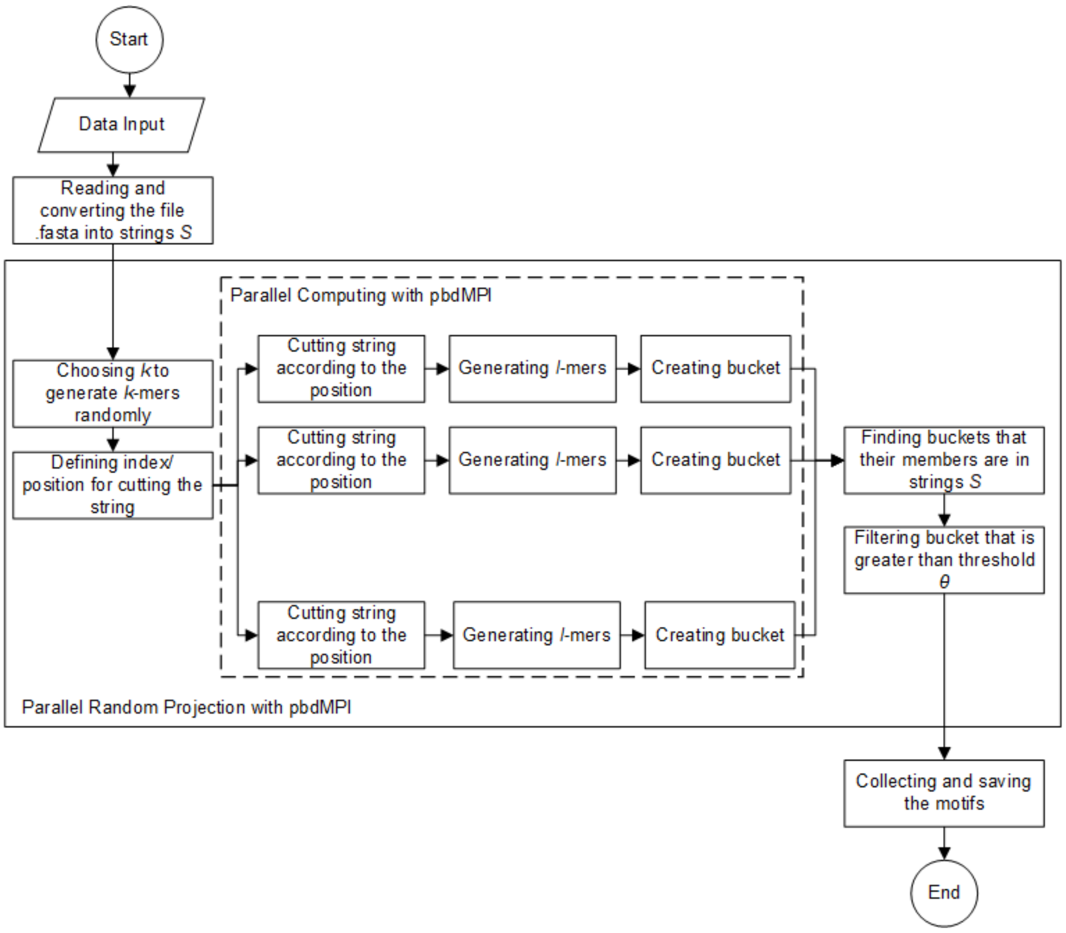

Start
Data Input
Reading and converting the file .fasta into strings *S*
Parallel Computing with pbdMPI
Choosing *k* to generate *k*-mers randomly
Defining index/ position for cutting the string
Cutting string according to the position
Generating *l*-mers
Creating bucket
Cutting string according to the position
Generating *l*-mers
Creating bucket
Cutting string according to the position
Generating *l*-mers
Creating bucket
Finding buckets that their members are in strings *S*
Filtering bucket that is greater than threshold *θ*
Parallel Random Projection with pbdMPI
Collecting and saving the motifs
End

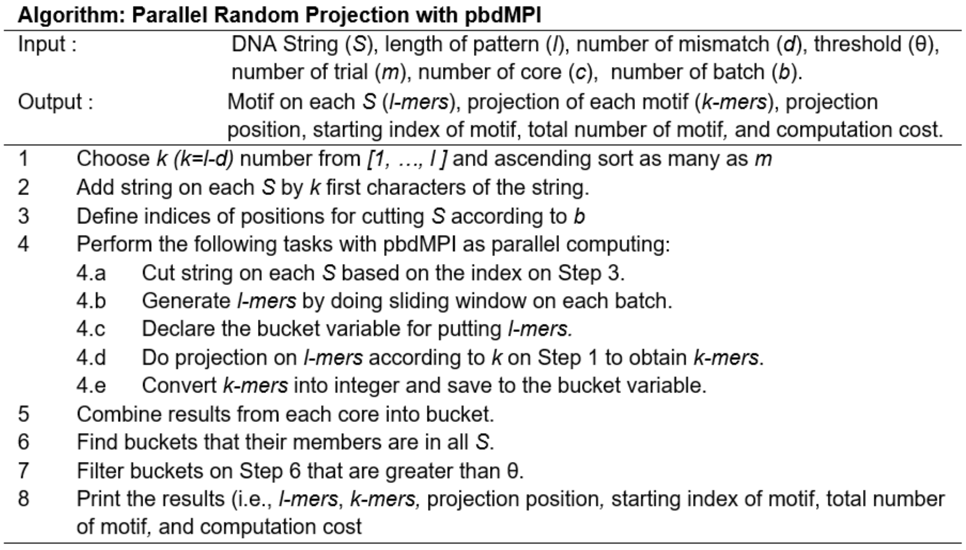
Figure 2. The computational model of parallel random projection with pbdMPI

According to Figure 3, it can be seen that besides supplying some parameters related to the RP algorithm, we need to input the number of cores and batches. Since the R programming language needs to load data into random access memory (RAM), we need to define the number of batches so that each batch just takes less than 20% of total memory capacity. Furthermore, actually Step 1 to 3 and Step 6 to 8 illustrated in Figure 3 are the same as the RP algorithm on the standalone mode. However, from Step 4 to 5 the tasks are conducted in parallel computing by using pbdMPI commands. An important part of these steps is a rule to divide the sequence into numbers of batches. Moreover, the rule should prevent all possible motif including the sequence even though it has been splitted into several batches. So, in this case we implement the (1) and (2):

$$index_s^i = \left(floor\left(\frac{L}{b}\right) * (i-1) \right) - (l-2), \tag{1}$$

$$index_e^i = floor\left(\frac{L}{b}\right) * i, \tag{2}$$

where $index_s^i$ and $index_e^i$ are starting and ending indices for cutting the batch of i. $L, b,$ and l are the length of sequence, number of batches, and length of pattern, respectively. It should be noted that the starting index starts from i=2. For example, it is give the sequence S=CAGTGACGTAATCA, and the length of pattern is 3. So, according to (1) and (2), we obtain the following batches: S_1=CAGT; S_2=GTGACG; and S_3=CGTAATCA. By following how the algorithm random projection generates *k-mers*, we obtain the following *k-mers* on all batches that are the same as *k-mers* on the sequence (without splitting into batches): CAG; AGT;GTG; TGA; GAC; ACG; CGT; GTA; TAA; AAT; ATC; and TCA. It means that even though the sequence has been splited and processed by different cores, the results of RP and parallel random projection are the same.



```
Algorithm: Parallel Random Projection with pbdMPI
Input :   DNA String (S), length of pattern (l), number of mismatch (d), threshold (θ),
          number of trial (m), number of core (c), number of batch (b).
Output :  Motif on each S (l-mers), projection of each motif (k-mers), projection
          position, starting index of motif, total number of motif, and computation cost.
1   Choose k (k=l-d) number from [1, …, l ] and ascending sort as many as m
2   Add string on each S by k first characters of the string.
3   Define indices of positions for cutting S according to b
4   Perform the following tasks with pbdMPI as parallel computing:
    4.a  Cut string on each S based on the index on Step 3.
    4.b  Generate l-mers by doing sliding window on each batch.
    4.c  Declare the bucket variable for putting l-mers.
    4.d  Do projection on l-mers according to k on Step 1 to obtain k-mers.
    4.e  Convert k-mers into integer and save to the bucket variable.
5   Combine results from each core into bucket.
6   Find buckets that their members are in all S.
7   Filter buckets on Step 6 that are greater than θ.
8   Print the results (i.e., l-mers, k-mers, projection position, starting index of motif, total number
    of motif, and computation cost
```

Figure 3. The pseudo code of parallel random projection with pbdMPI

4. Experimental Study

4.1. Data Gathering

The data used in this study obtained from research in [17]. To download the data can be through the site of University of Washington Computer Science and Engineering on page http://bio.cs.washington.edu/research/download. In total, there are 52 data sets of DNA sequences derived from four species, 6 of which are derived from the Drosophila melanogaster sequence, 26 data derived from human sequences, 12 data derived from rat sequences and 8 other data derived from the Saccharomyces cerevisiae sequence. In each data file there are several sequences that number between 1 to 35 sequences. Then, every sequence that resides on the file has a variable length ranging from 500 to 3000 base pairs.

In this case, we only consider to use four datasets as follows: the dm01r.fasta and dm05r.fasta files that are DNA sequences of Drosophila melanogaster, then hm01r.fasta derived from the human sequence, and muso4r.fasta which is the rat DNA sequence as the input data. The dm01r.fasta file contains 4 DNA sequences with the total length of sequence is 6000, while the dm05r.fasta file consists of 3 DNA sequence with the length of 7500. The hm01r.fasta and mus04r.fasta files have the DNA sequence length of 36000 and 7000, respectively.

4.2. Experimental Design

In this study we conduct two simulations: standalone and parallel computing (i.e., multicore) modes. Each group will use all data as mentioned previously: dm01r.fasta, dm05r.fasta, hm01r.fasta, and muso4r.fasta. Furthermore, in accordance with the algorithm, some parameters should be assigned, as follows: the length of motif and mismatches (*l, d*), threshold values (*θ*), and number of repetitions in each simulation (*m*). Specific variables (i.e., number of batches (*b*) and cores (*c*)) on parallel computing with pbdMPI are also assigned. Total experiments conducted for both scenarios are 1560 experiments (i.e., 120 times for standalone and 1440 times for parallel computing) by assigning all possibilities of the combinations of the following parameter values: (l, d)={(6,2), (7,2), (8,3)}, *θ*={3,4}, *m*={1,2,3,4,5}, *b*={10,50}, and *c*={1,2,…,6}.

5. Results and Analysis

Since the limited space, in this section we illustrate the results and their analysis for a particular dataset only. For example, on the standalone mode, a comparison of the number of motifs found according to *m*, *θ*, and (*l, d*) on the dm01r dataset is shown in Figure 4. It can be seen that the higher numbers of mismatch makes the higher of numbers of motifs.

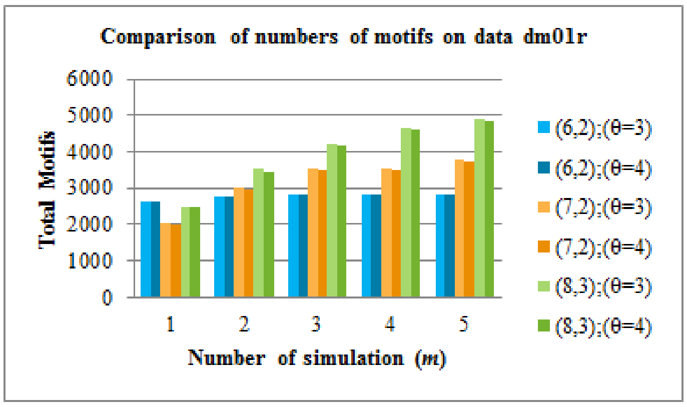


Figure 4. The comparison of the numbers of motifs found on the dm01r dataset

Furthermore, on the standalone mode, we can compare the computational cost with length of DNA sequence on the different (*l, d*) and *θ* as shown in Figure 5. It is obvious that the longer length of DNA sequence takes the higher computation cost. It should be noted that these lengths also represent the datasets used in the experiments, such as the dm01r has the length of 6000.

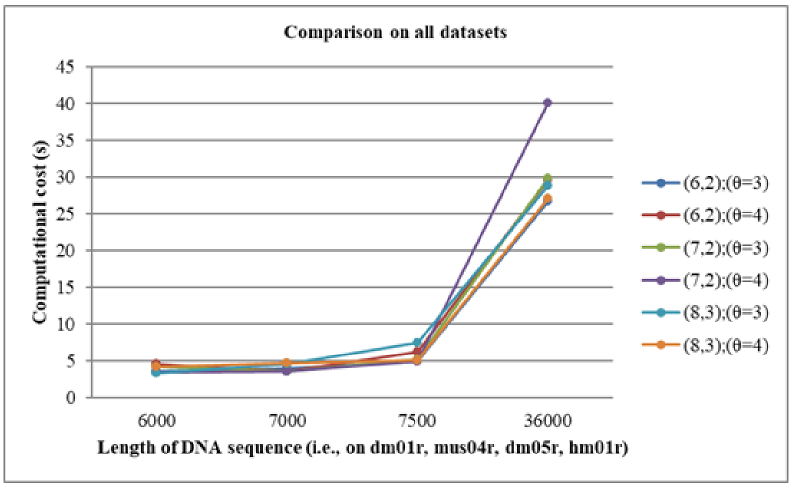


Figure 5. The comparison between the computational cost and datasets/length of datasets

On the parallel computing mode, Figure 6 shows that the comparison between the computational costs and numbers of cores when we used the dm01r dataset on (*l, d*)=(6.2), *θ*=3, and *b*=10. It can be seen that the proposed model has been successful since in general speaking the computation cost can be reduced by adding the number of cores.

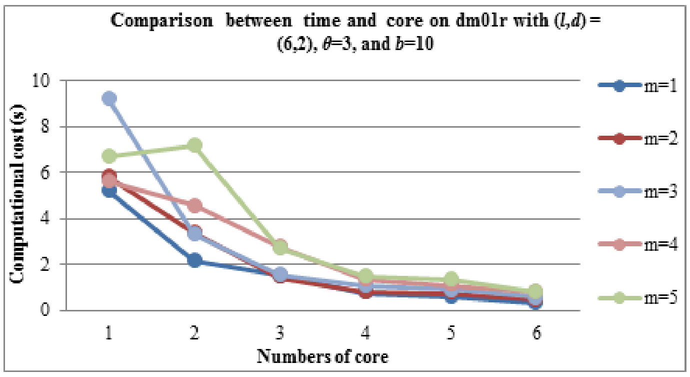


Figure 6. The comparison between computational cost and number of core on the dm01r dataset

To ensure the analysis, Figure 7 explains a comparison between computational cost and number of core on different (*l, d*) and m and the same *θ* (i.e., 3), and *b* (i.e., 10). It can be seen that the computational time with stand alone mode (i.e., *c*=1) at (*l, d*)=(8.3) with *m*=5 took 26.98 seconds while on the number of core of 2 the computation only took 6.3 seconds. It means that the computational time on stand alone needs four times longer than using 2 cores. Moreover, the standalone mode took more than ten times compared with parallel computing using 3 cores (i.e., 2.52 seconds). Using 6 cores, the computation can be faster around 34 times compared with the standalone mode. So, now it is obvious that the proposed model is much faster than the standalone mode.

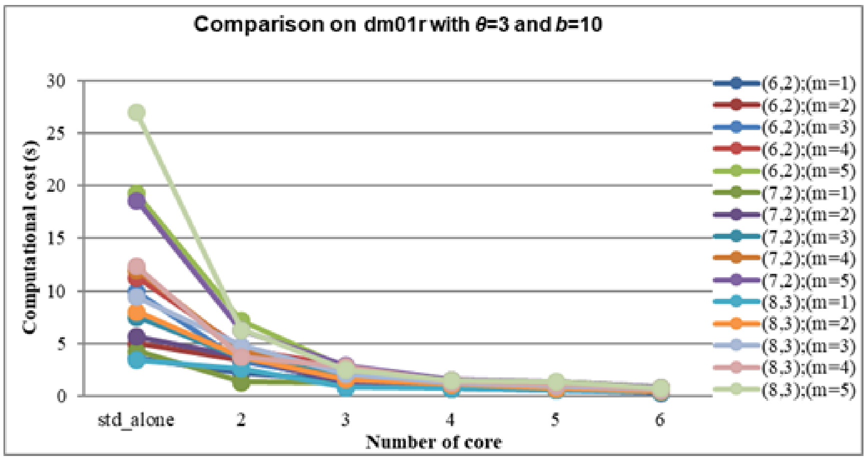


Figure 7. The comparison on dm01r with *θ*=3 and *b*=10

We also compared computational time gained from experimental results on the previous research [1] even though there are different data on the file dm01r and mus04r. The number of DNA sequences contained in the file dm01r is 4 with the length of 1500 for each sequence while in the research [1] the dataset contains 5 DNA sequences. In the file mus04r the number of

DNA sequences used in this experiment is 7 sequences with the length of each sequence is 1000 while only 6 sequences were used by the previous research. The comparison can be seen in Table 1. It can be seen that all experiments conducted in this research are faster than the study in [1]. It should be noted that the research conducted by [1] was performed in standalone mode.

Table 1. Comparison with the Other Research [1]

File	Number of sequence	Length of each sequence	(*l*,*d*)	Time in [1] (s)	Time using 6 cores in this research (s)
			(6,2)	1.81	0.32
dm01r	4	1500	(7,2)	5.804	0.45
			(8,3)	39.55	0.42
			(6,2)	2.574	0.48
dm05r	3	2500	(7,2)	7.268	0.51
			(8,3)	52.537	0.52
			(6,2)	2.73	1.75
hm01r	18	2000	(7,2)	7.708	1.65
			(8,3)	50.135	1.71
			(6,2)	1.4	0.52
mus04r	7	1000	(7,2)	5.08	0.5
			(8,3)	35.45	0.52

6. Conclusion

The main contributions of this research are as follows (i) to propose the computational model for modifying the random projection algorithm, called parallel random projection, for dealing with planted motif search by utilizing R high performance computing (i.e., the pbdMPI package) and (ii) to implement the proposed model and then validate it for finding motifs on DNA sequences. According to the experiments, we can state that the proposed model are able to reduce the computational cost significantly. Moreover, a comparison with the previous study has been done, and it shown that the proposal produced better results in the term of computational cost.

In the future, we have a plan to improve the model by using Big Data platform, such as by using the programming model of MapReduce on Apache Hadoop [18] and Resilient Distributed Datasets on Apache Spark [19]. Moreover, the different tools for utilizing parallel computing, e.g., the foreach package [20], can be used as the study in [21]. Different tasks in the related research to bioinformatics can be applied to test the proposed model as well, such as prediction on cáncer [22], kidney disease [23], and sleep disorder [24]. Additionally, another method that can be implemented for dealing with this research is Knuth Morris Pratt [25, 26].

References

[1] Ashraf F Bin, Abir AI, Salekin MS, et al. *RPPMD (Randomly projected possible motif discovery): An efficient bucketing method for finding DNA planted Motif*. In: *ECCE 2017* - International Conference on Electrical, Computer and Communication Engineering. 2017: 509–513.
[2] Reece JB, Urry LA, Cain ML, et al. *Campbell Biology, 10th*. Pearson: Boston, MA, 2014.
[3] Aluru S. *Handbook of computational molecular biology*. Chapman and Hall/CRC, 2005.
[4] Rajasekaran S, Balla S, Huang C-H, et al. High-performance exact algorithms for motif search. *Journal of clinical monitoring and computing*. 2005; 19(4–5): 319–328.
[5] Pal S, Rajasekaran S. *Improved algorithms for finding edit distance based motifs. In: Bioinformatics and Biomedicine (BIBM), 2015* IEEE International Conference on. 2015: 537–542.
[6] Martinez HM. An efficient method for finding repeats in molecular sequences. *Nucleic acids research*. 1983; 11(13): 4629–4634.
[7] Nicolae M, Rajasekaran S. Efficient sequential and parallel algorithms for planted motif search. *BMC bioinformatics*. 2014; 15(1): 34.
[8] Buhler J, Tompa M. Finding motifs using random projections. *Journal of computational biology*. 2002; 9(2): 225–242.
[9] Jones NC, Pevzner PA, Pevzner PA. *An introduction to bioinformatics algorithms*. MIT press, 2004.
[10] Ihaka R, Gentleman R. R: a language for data analysis and graphics. *Journal of computational and graphical statistics*. 1996; 5(3): 299–314.
[11] Ostrouchov G, Chen W-C, Schmidt D, et al. Programming with big data in R. *URL http://r-pbd org*.

[12] Clemente JB, Adorna HN. Some Improvements of Parallel Random Projection for Finding Planted (l, d)-Motifs. In: *Theory and Practice of Computation*. Springer, 2013: 64–81.
[13] Hussein AM, Rashid NA, Abdulah R. Parallelisation of maximal patterns finding algorithm in biological sequences. In: *Computer and Information Sciences (ICCOINS), 2016 3rd International Conference on*. 2016: 227–232.
[14] Mohanty S, Sahoo B, Balouki A, et al. Enhanced Planted (ℓ, D) Motif Search Prune Algorithm for Parallel Environment. *Journal of Theoretical and Applied Information Technology*. 2016; 89(2): 296–306.
[15] Jun L, Zhuo Z, Juan M, et al. Multi-pattern Matching Methods Based on Numerical Computation. *Indonesian Journal of Electrical Engineering and Computer Science*. 2013; 11(3): 1497–1505.
[16] He J, Guo H, Hu D. A Neighbor-finding Algorithm Involving the Application of SNAM in Binary-Image Representation. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*. 2015; 13(4): 1319–1329.
[17] Tompa M, Li N, Bailey TL, et al. Assessing computational tools for the discovery of transcription factor binding sites. *Nature biotechnology*. 2005; 23(1): 137.
[18] White T. *Hadoop: The definitive guide*. ‘ O’Reilly Media, Inc.’, 2012.
[19] Zaharia M, Xin RS, Wendell P, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*. 2016; 59(11): 56–65.
[20] Analytics R, Weston S. foreach: Foreach looping construct for R. *R package version*. 2013; 1(1): 2013.
[21] Riza LS, Utama JA, Putra SM, et al. Parallel Exponential Smoothing Using the Bootstrap Method in R for Forecasting Asteroid’s Orbital Elements. *Pertanika Journal of Science & Technology*. 2018; 26(1): 441–462.
[22] Michiels S, Koscielny S, Hill C. Prediction of cancer outcome with microarrays: a multiple random validation strategy. *The Lancet*. 2005; 365(9458): 488–492.
[23] Alasker H, Alharkan S, Alharkan W, et al. *Detection of kidney disease using various intelligent classifiers*. In: Science in Information Technology (ICSITech), 2017 3rd International Conference on. Bandung, 2017: 681–684.
[24] Riza LS, Pradini M, Rahman EF, et al. *An Expert System for Diagnosis of Sleep Disorder Using Fuzzy Rule-Based Classification Systems*. In: IOP Conference Series: Materials Science and Engineering. Bandung, Indonesia, 2017: 12011.
[25] Knuth DE, Morris Jr JH, Pratt VR. Fast pattern matching in strings. *SIAM journal on computing*. 1977; 6(2): 323–350.
[26] Riza LS, Firmansyah MI, Siregar H, et al. Determining Strategies on Playing Badminton using the Knuth-Morris-Pratt Algorithm. *TELKOMNIKA Telecommunication Computing Electronics and Control*. 2018; 16(6):2763-2770.