Asian Journal of Information Technology 15 (23): 4835-4843, 2016
ISSN: 1682-3915

Improved Exact Parallel Algorithm for Planted (l, d) Motif Search

Satarupa Mohanty and Biswajit Sahoo
School of Computer Engineering, KIIT University, Bhubaneswar, India

Abstract: Motif search in computational biology is a most challenging problem. This plays a crucial role in gene finding and understanding the gene regulation relationship. In this study, a new efficient algorithm is proposed for the (l, d) motif search problem to find all string of length l which present in each of the input string with d mismatches. The method is based on 2 key aspects. First, a group of 3 l-mers of close proximity is processed efficaciously to generate the common d-neighborhood and second the data structure bit vector is used which simplifies the process of making the union and intersection of the common d-neighborhood. The proposed approach can be considered to be a hybrid one, as it integrates the existing algorithm with the novel ideas of common d-neighborhood generation to achieve better running time. Moreover, a parallel version of proposed method is also presented which runs on 4 SMP cluster systems with each of 2.4 GHz Intel Pentium-IV having 16 GB ram running under Red Hat Linux. The experimental result shows that the proposed algorithm is linearly scalable with the number of processors.

Key words: Planted motif search, Symmetric Multi Processor (SMP), Message Passing Interface (MPI), bit map vector, string algorithm

INTRODUCTION

Pattern reorganization in biological sequences is a major problem as it led to numerous solutions in the biological domain. For example, motifs are such patterns found in biological sequences that have applications in genetic probe design, PCR primer design, discovering potential drug targets, finding unbiased consensus of a protein family, antisense drug design, creating diagnostic probes. In literature, several variants of this motif search problem have been suggested. Among those the Planted (l, d) Motif Search (PMS) problem is considered here. PMS is defined as follows: inputs to the PMS are n biological sequences of length m each, 2 positive integer's l and d. The objective is to extract strings M of length l such that any such string M is present in all n sequences with at most d mismatches. Formally, the problem can be defined as follows.

Definition 1: Given a set of n sequences $s:=\{s_i\}_{i=1}^{n}$ over an alphabet Σ = {A, T, C, G} with $|s_i|$ = m and integers l , d with $0 \le d < l < m$, the PMS resolves the task of identifying the (l, d) Motif x with |x| = l such that x_i is a substring of s_i of length l and x differs from x_i in at most d places for i = 1, ..., n.

In the PMS problem, standard values of n and m considered are 20 and 600. Different combinations of l and d give rise to different occurrences of PMS. The occurrences where the d value is large in comparison to the l value, are called weak occurrences and are difficult to resolve. For example, occurrences (13, 4), (15, 5), (17, 6), (18, 6), (19, 7), (21, 8), (23, 9), etc. are well-known weak occurrences. Owing to its importance, development of the efficient algorithm for PMS is now one operational interest in bioinformatics. There are 2 categories of PMS algorithms, namely, approximation and exact algorithms on the basis of heuristic search and exhaustive enumeration search respectively. Generally, PMS approximation, algorithms tend to be faster and more popular than exact algorithms but they are not guaranteed to give the correct motif always. Mostly approximation algorithms exploit heuristics search such as expectation optimization, local search, Gibbs sampling, etc., examples of approximation algorithm are Gibbs (Lawrence *et al*., 1993), MEME, random projection (Rocke and Tompa, 1998), Consensus (Herts and Stormo 1999), Winnower (Pevzner, and Sze, 2000), Projection (Buhler and Tompa, 2002), Multiprofiler (Keich and Pevzner, 2002), Pattern Branching (Price *et al*., 2003) and Vine (Huang *et al*., 2011). Among these Gibbs and MEME is the simple approach which uses certain groups of start sites initially to avoid local optimum and Vine (Huang *et al*., 2011) is the recent heuristic polynomial time algorithm which is based on winnower (Pevzner and Sze, 2000).

In this study more attention is on the exact algorithms. Exact algorithms are always guaranteed to find

Corresponding Author: Satarupa Mohanty, School of Computer Engineering, KIIT University, Bhubaneswar, India

all existing motifs in spite of the worst case exponential running time by virtue of its NP-hard nature (Evans and Smith, 2003). The exact algorithms are classified into sample driven and pattern driven approaches. To obtain the desire motif the sample driven approach explore the d-neighborhoods of all possible (m-l+1) n l-mers and the pattern driven approach enumerates all Σ^l possible patterns. The sample driven part is often constrained by space complexity. On the contrary, the pattern driven approach objective is to reduce the candidate motif through numerous approach. Roughly there are four approaches present in literature to solve the exact algorithms. The first approach builds a graph by considering its vertices as the length l substring of the input sequences and its edges as the connector between two vertices if they are owned by different input sequence with Hamming distance within 2d. Then, it transforms the problem of motif search into a problem of finding a clique of size n where n represents the number of input sequences. Examples are DPCFG (Yang and Rajapakse, 2004), Rec motif (Sun *et al*., 2010a, b), List and Tree motif (Sun *et al*., 2011).

The second approach utilises the suffix tree introduced by Sagot (1998). For all the given sequences he builds a generalized suffix tree and then uses the suffix tree to "spell" all the motifs. Examples are Speller (Sagot, 1998), Weeder (Pavesi *et al*., 2001), MITRA (Eskin and Pevzner 2002), CENSUS (Evans and Smith, 2003), RISOTTO (Pisanti *et al*., 2006). RISOTTO is the fastest among the family of suffix tree algorithm but with the increase in the motif length, its performance degrades significantly. The third approach initially computes the candidate motif from some input strings and then searches for its feasibility in the rest of the strings. Examples are voting (Chin and Leung, 2005), PMS1-PMS3 (Rajasekaran *et al*., 2005), algorithm of (Sze and Zhao, 2007), PMSi and PMSP (Davila *et al*., 2006), PMS4 (Rajasekaran and Dinh, 2011), PMS5 (Dinh *et al*., 2011), PMS6 (Bandyopadhyay *et al*., 2014), Pair motif (Yu *et al*., 2012). The fourth approach builds upon the search tree method which choose a candidate l-mer from the first input string and then modifies its character one by one. Since the mismatch between the candidate and the motif is at most d, the search tree of depth d is traversed to arrive at the motif. Examples include PMS prune, Pampa (Davila *et al*., 2007 a, b), q PMS prunel and qPMS7 (Dinh *et al*., 2012). Some algorithm utilizes an efficient combination of sample driven and pattern driven approach such as the algorithm of (Sze *et al*., 2004), PMS8 (Nicolae and Rajasekaran, 2014) and qPMS9 (Nicolae and Rajasekaran, 2015). Many of the above mentioned efficient exact algorithms (PMS5 (Dinh *et al*., 2011), PMS6 (Bandyopadhyay *et al*., 2014), qPMS7 (Dinh *et al*., 2012)) compute common d-neighborhood of a group of three l-mers coming from three input sequences and match with the remaining to certify or reject it as a motif. For the pre-processing they use the Integer Linear Programming (ILP) whose number of instances increases with the increase in the number of iteration and also a lookup table is required to be populated repeatedly and to be searched for determining the common neighbors. This pre-processing leads to a plentiful memory requirement for storing the lookup tables and the outcomes of all possible integer linear programs. To efficiently solve the PMS problem, a combination of pattern-driven and sample driven approach is proposed here. The PMS6 (Bandyopadhyay *et al*., 2014) is regarded as the basis of our approach. The motivation of the proposed approach by the observation is as follows:

- Elimination of pre-processing steps
- The volume of candidate motif common to two l-mers or three l-mers are greatly rely upon the distances between the l-mers. So the selection of multiple l-mers should be done meticulously from the input sequence to restrict the total candidate volume
- A novel approach of common d-neighborhood generation three l-mer is proposed to filter out surplus candidate verification
- Use of bit vector to efficiently handle the process of union and intersection of the candidate motif

In accordance with the above techniques we implement the parallel version of this approach which exhibits that the proposed one exceed the PMS5 (Dinh *et al*., 2011) and PMS 6 (Bandyopadhyay *et al*., 2014) in a several ways and also is a competent with recent fastest algorithm as PMS 8 (Nicolae and Rajasekaran, 2014) and qPMS 9 (Nicolae and Rajasekaran, 2015).

MATERIALS AND METHODS

We use same notations and definitions as in (Bandyopadhyay *et al*., 2014) as follows. A string of length l is an l-mer. The l-mer x of any string s, symbolized as $x \in_l s$ is a length l substring of s. The hamming distance of 2 l-mers x and y, $d_H(x, y)$ is the number of mismatch between them. The 2 l-mers x and y agreeing $d_H(x, y) \leq 2d$ are the matter of consideration because the 2 instances of the same motif cannot differ by more than 2d. For a given string s and l-mer x, the $C(x, s)$ symbolizes the set of $y \in_l s$ satisfying $d_H(x, y) \leq 2d$. Given a set of strings $S = \{s_1, ..., s_n\}$, the $M_{l,d}(S)$ stand for the (l, d) motif of S.

The proposed algorithm is based upon a combination of sample driven and pattern driven approaches. Initially in sample driven part, we select a triplet of l-mers from string s_1, s_{2k}, s_{2k+1}, for k = 1 to n/2 , satisfying some filtering condition. Then in pattern driven part we compute the common d-neighborhoods of the triplet by sampling characters position wise. Our method is mainly composed of three steps:

- Selecting triplets for each l-mer x of s_1, select an l-mer y from C (x, s_{2k}) and an l-mer z from C (x, s_{2k+1}) and form a triplet of l-mers as (x, y, z)
- Finding common d-neighborhood we take one candidate common neighbor of the triplet (x, y, z) and then apply a recursive procedure on it to find the set of all common d-neighborhoods
- Making union and intersection we apply a series of intersection and union operations on generated common d-neighborhoods to obtain the motif Ml, d (S). To address the union and intersection operation comfortably we use the bit vector at an expense of some additional space

The problem can be considered to have independent (m- l+1) number of sub problems, one for each l-mer of the sequence s_1. For every subproblem, the input is the l-mers of sequence s_1 and the input sequences s_2-s_n. This shows the problem is embarrassingly parallel without any inter process communication besides the input to all processors. We exhibit two levels of parallelism, the outer level parallelism using MPI and inner level parallelism using threads. In the outer level we considered the processor with rank 0 as the scheduler with rest as workers. The scheduler broadcast the required input to all workers including itself and solve the sub problems simultaneously with other workers to avoid one processor only to be busy with scheduling. At the end, the scheduler receives the motifs from all workers and makes the union with its own motifs. The scheduler finally produces the result. The execution time for each processor is very sensitive to the volume of neighborhoods of various triplets and hence, some processor ends up starving while the others are busy. In the inner level, multiple threads serve simultaneously on different l-mers x of s_1. The threads cooperate to find common neighborhood corresponds to every l-mer of s_1. Based on the above 3 steps the proposed parallel approach is depicted in Algorithm 1.

Algorithm 1: Parallel planted (l, d) motif search
Input: S = {s_1, s_2, ..., s_n}, l, d
Output: the (l, d) motif set $M_{l,d}(S)$

```
1  M_{l,d}(S) = Ø
2  for each X∈_1 s_1 do parallel using process
   Par Begin
3    M = Ø
4    for k = 1[n-1/2] to  do parallel using thread
       for each y ∈ C(x, s_2k) and z ∈  C(x, s_2k+1) do
         if (k = 1) then
           M = M∪find_common (x, y, z)
         else
           M = M∩find_common (x, y, z)
       end for
5    end for  // Motif of different thread
6    M_{l,d} = M_{l,d}∪M
7  Par end
8  end for
9  M_{l,d}(S) = Um_{i,d} //Motif of different process
10 Output M_{l,d}(S)
```

Line 1 assigns an empty set to the (l, d) motif $M_{l,d}$ in the beginning. Line 2-13 find the (l, d) Motif corresponds to every l-mer of string s_1 in parallel using MPI. Line 4-11 find the common neighbor of the selected triplets using the procedure find_common in parallel using thread. Line 9 intersects the common neighbors of the selected triplets of multiple thread to find the (l, d) motif corresponds to a single l-mer of s_1. Line 12 makes the union of (l, d) motif of different l-mer of s_1. Finally, line 13 merges the (l, d) motif of different processes into the (l, d) motif of the given input sequences.

Step 1 (selecting triplets): We select the triplet in a decisive way. We do not form the triplet just by considering every l-mer of different sequences but rather we apply one filtering rule. The filtering rule says that if two l-mers x and y have a common d-neighbor Z such that $d_H(x,z) \leq d$ and $d_H(y,z) \leq d$ then $d_H(x,y) \leq d+d = 2d$. The filtering rule employs the criterion that the distance of 2 occurrences of the same motif must not be more than 2d. So we only consider the l-mers x' of s_i which is at hamming distance $\leq 2d$ from l-mer x of s_1 (or formally $x' \in C(x, s_1)$).

Step 2 (finding common d-neighborhood): For every triplet of l-mers we find the common d-neighborhoods using the algorithm find-common. Given a triplet of l-mers x_1-x_3 an l-mer q is common d-neighbor of the triplet iff:

$$\max_{i=1}^{3} d_H(x_i, q) \leq d$$

The triplet of l-mers x_1, x_2, x_3 can be represented as a 3×l character matrix where the columns type can be defined as follows. Type P_0 are the columns where $x_1[I] = x_2[I] = x_3[I]$. Type P_1 are the columns where $x_1[I] \neq x_2[I] = x_3[I]$. Type P_2 are the columns where $x_2[I] \neq x_1[I] = x_3[I]$. Type P_3 are the columns where $x_3[I] \neq x_2[I] = x_1[i]$. Type P_4 are the columns where $x_1[I] \neq x_2[i] \neq x_3[I]$. We perform a

preprocessing to generate a candidate common l-mer q as follows. q [i] = x_1 [i] for i ∈ Type P_0 or i ∈ type P_4, q [i] = x_2 [i] for i ∈ Type P_1, q [i] = x_3 [i] for i ∈ Type P_2, q [i] = x_2[i] for i ∈ Type P_3. We input the candidate motif q and the distance D as 'd' initially to the recursive procedure find-common.The algorithm successively decreases D by one, till D becomes <0 (Algorithm 2).

Algorithm 2 (Find-common (x_1-x_3):

```
Input: 3 l-mers x1-x3
Output: All common neighborhoods
do Preprocessing to produce candidate Motif q
find_common (x1, x2, x3, q, D)
  if max_{i=1}^{3} d_H(x_i,q) > d + D) then return l-mer q
If max_{i=1}^{3} d_H(x_i,q) > d + D) then return
  if (D < 0) then return
  for each i ∈{1, 2, 3} and d_H(q, x_i)>d do
    P = {k| q[k] ≠ x_i[k]}
    for each set pos such that pos⊆ P and|pos| =d+1
      for all j ∈ pos do
        q_new = q
        for α∈Σ and α ≠ q[j]
          q_new[j] = α
          return (find_common(x1, x2, x3, q_new, D-1)
        end for
      end for
    end for
  end for
```

Line 1 report the l-mer q as a common neighbor when the recursive procedure satisfies the inequality as:

$$\max_{i=1}^{3} d_H(x_i,q) \le d$$

Line 2 and 3 show the invalid attempt of the candidate l-mer. Line 4 to line 10 tries to construct a new candidate l-mer q_{new} to reduce its distance from l -mer x_i for i ∈ {1, 2, 3} when the candidate l-mer q is not a solution but there is a possibility of getting the common l-mer. To create the new l-mer q_{new} consider all position k where q [k] ≠ x_i [k] for i∈{1, 2, 3} such that d_H (S, x_i)>d and successively generate sub cases of d+1 positions to alter the positions of q by character α where α∈Σ and α ≠ q [j]. Taking advantage of this recursive algorithm we find all solution to a given instance in a reasonable time.

Step 3 (making union and intersection): We use bit vector approach for its simple way of making the union and intersection operation and its amenability to parallelization. We allocate three bit vectors of size 4^l bits each. The basic operation on bit vector is divided into 3 phases. In line 7 and 9 of algorithm1 for every common d-neighbor of the triplet we set the corresponding bit in the bit vector M. The index in accordance to a l-mer can be derived by substituting the characters {A, G, C, T} by 2 bits {00, 01, 10, 11}. For instance of a 5 l-mer ACTTG, the index is the integer value of 0010111101. In line 9, we do the intersection of common l-mers to find common d-neighbor correspond to a particular l-mer x of sequence s_1. In line 12 and 14, we do the union operation to generate motif of multiple thread and multiple processes, respectively. The union and intersection are simple bitwise union and intersection operation and can be easily parallelizable. The bit vector has high memory requirement with the increase in the size of l-mers. As a solution for higher l-mer sizes for l≥17, we use a cumulative approach where we initially find the set of motifs of (l-1, d) instance with their distances from every sequences. Then successively we find the solution for the motif instance (l, d) in O (n) time. For simplification let M is a (l-1, d) Motif instance with (p_2, p_3, ..., p_n) and (d_2, d_3, ..., d_n) as the location of d-neighbors and their distances from the (n-1) sequences, respectively. We can state that the string M|C where C∈Σ and '|' is an append operation is a (l, d) motif or has a d-neighbor in every sequence s_i, if it satisfies one of the following conditions: residue at position p_i + 1 is C or s_i [p_i+ 1] = C, or the new distances d_i<d. For each motif M we append the C∈Σ and check whether p|A, p|G, p|C, p|T is a motif or not by applying the above conditions. Thus to find (l, d) motif for higher l, i.e., l≥17 we first find (16, d) motifs and then successively apply the above logic to find any motif of higher sizes.

Implementation of proposed algorithm: We have employed a Client-Server (C/S) architecture in our LAN system to implement this algorithm, i.e. one server and multiple clients. The server coordinates and distributes the PMS tasks to the clients and takes the responsibility of their synchronization as showed in Fig. 1. The clients receive the task from the server and execute it in a parallel manner. We use the SSh key approach to avoid the password requirement during the interaction among master system and clients (slave) system for the execution of the program. The easy and fast communication between systems can be established through this.

Server designing: The server has the following functions according to its design.

Task distribution: In the consideration of simultaneous work of clients, the server divides the task into smaller subtasks depending upon the number of files or sequences present and distributes the task to the clients for the uniform computation. Clients return the produced results at the end.

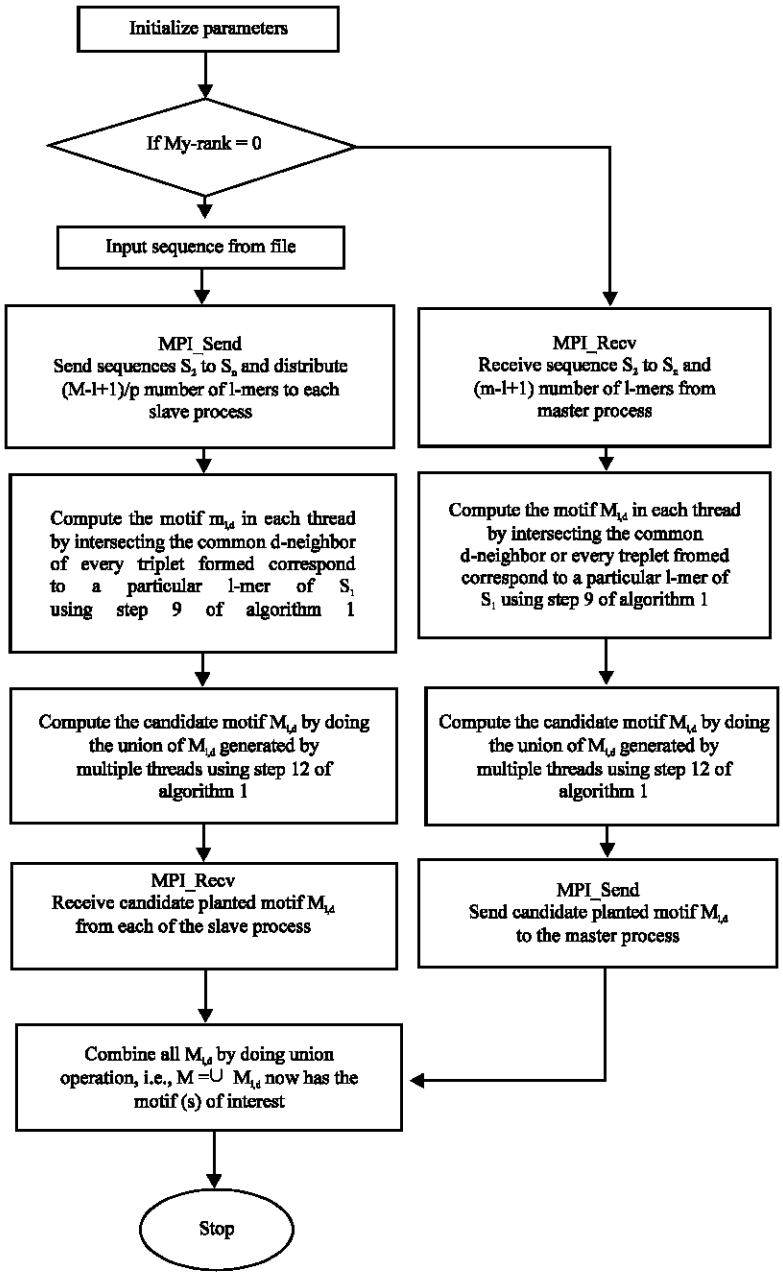

Fig. 1: Flow chart of all message passing among master and slaves

Database sharing: All the bioinformatics information is stored in the main database on the server. The synchronization of the client is monitored by the server which is just to have the one to one correspondence between the modules. The 'Start' message is sent by the server after the distribution of the tasks so that they can begin to work together. After the completion of work, the client will send an 'end' message and result to the server to say the completion of the task.

Planted (l, d) Motif: The server begins to calculate the planted (l, d) motif after receiving the 'end' messages from all the clients. The server is configured with Redhat enterprise version-5 OS.

Client designing: Clients complete the task of receiving DNA sequences into their database and computing the planted (l, d) motif from the subsets of sequences under the control of the server. For this, we have another

database on the clients which act as a secondary database. In the client side, the download of information starts from the server after receiving tasks from it and it stores those on the secondary storage by which the time delay due to successive accessing the primary storage can be solved. After completion of the current task, the output is submitted to the server by clients with one 'End' message to the server. Clients are configured with Redhat enterprise version-5 OS.

RESULTS AND DISCUSSION

Experimental result and performance analysis: The algorithm that we proposed is implemented in the C language using the MPI library, Pthread library. The experiment is done in an environment where there are four node clusters with each of 2.4 GHz Intel Pentium-IV processor having 16 GB RAM running under Red Hat Linux. One Giga-bit Ethernet switch we have used to connect the nodes. Our parallel algorithm has been examined with all four nodes and the starting and ending execution time has been evaluated with the wall clock time. The running time here is the sum of its execution time, its communication delay and time to get input data from a file. The algorithm allocates the sequence pairs based on the number of Symmetric Multiprocessor (SMP) nodes available. Due to our LAN connection the distance between server and client is very short, the distance of transferring data or result between the server and the clients can be neglected as.

Our computational experiment shows the effectiveness of the proposed algorithm. We use fixed number mutation for generating random datasets of n sequences of length m each so that each character of Σ appears with uniform probability. Then, in the same way, we generate the l length motif by mutating in just d places and plant in all n sequences randomly. We use the number of sequences n as 20 and the length of sequence m as 600. To show that our proposed algorithm runs faster, the execution time for challenging values of l and d are compared against some of the well-known fastest exact algorithms in Table1. In Table1, the symbol '-' means either the algorithm take a longer time to execute or takes more memory (Table 1).

The first set of experiments is intended to observe how lengths of the planted motif affect the performance of our parallel algorithm with respect to some of the existing PMS algorithms. Figure 2 shows the speed up of the proposed algorithm with respect to some of the recent algorithms. It is found in all cases that correct planted motifs are found using the proposed parallel algorithm successfully. The next set of experiments investigates how the processing time is influenced by varying Hamming distance for the distinctive length of motifs. Table 2 shows the behaviour of the proposed algorithm and its parallel version for various values of (l, d) on a

Table 1: Time comparison of different algorithms on challenging instances

Instance algorithms/(l, d) motif	(13, 4)	(15, 5)	(17, 6)	(19, 7)	(21, 8)	(23, 9)
Proposed algorithm	7 sec	28 sec	1.33 m	10.47 m	41.5 m	1.56 h
qPMS9	6 sec	34 sec	2.7 m	13.4 m	45.4 m	2.26 h
PMS8	7 sec	48 sec	5.2 m	26.6 m	1.64 h	5.48 h
qPMS7	29 sec	2.1 m	10.3 m	54.6 m	4.87 h	27.09 h
PMS6	22 sec	75 sec	6.72 m	22.75 m	2.25 h	19.19 h
PMS5	39 sec	130 sec	11.35 m	40.38 m	4.96 h	40.99 h
PMSprune	53 sec	9 m	69 m	9.10 h	-	-

Table 2: Time comparison of proposed algorithm for different (l, d) instances

			Processing time for parallel algorithm	
Length of the motif (l)	Hamming distance (d)	Processing time for prposed algorithm on single CPU	Two CPU	Four CPU
15	5	28 sec	25 sec	19 sec
	4	14 sec	11 sec	4.5 sec
	3	4 sec	3	2.7 sec
17	6	1.33 m	56 sec	43 sec
	5	69 sec	43 sec	34 sec
	4	34 sec	22 sec	15 sec
19	7	10.47 m	7.54 m	4.56 m
	6	7.45 m	4.47 m	3.25 m
	5	3.52 m	2.32 m	1.39 m
21	7	34.45 m	22.42 m	17.43 m
	8	41.5 m	34.33 m	21.45 m
	9	54.32 m	47.44 m	27.32 m
23	8	1.04 h	0.54 h	0.35 h
	9	1.56 h	1.04 h	0.56 h
	10	2.34h	1.45h	0.55h

multiple number of node cluster. In the parallel version of the proposed algorithm, the work to be carried out is distributed uniformly.

Our parallel algorithm is executed by creating 2 numbers of processes on a 2 node cluster and 4 numbers of processes on 4 node cluster. This distributes one process to each node. Curves in Fig. 3 indicates that the processing time has linear growth as the number of processors increase. The next experiment investigates how the processing time is affected by varying Hamming distance for the length of the motif 15, 17, 19, 21, 23 as showed in Fig. 4-8, respectively. With the increasing

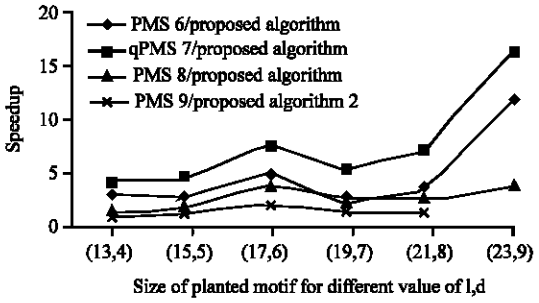

Fig. 2: Speedup of the proposed algorithm over the well known existing algorithms as a function of (l, d) motifs for different values of l and d

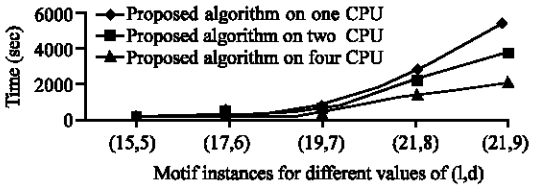

Fig. 3: Running time comparison of proposed algorithm on different number of processor(s) (1P, 2P,4P) as a function of (l, d)

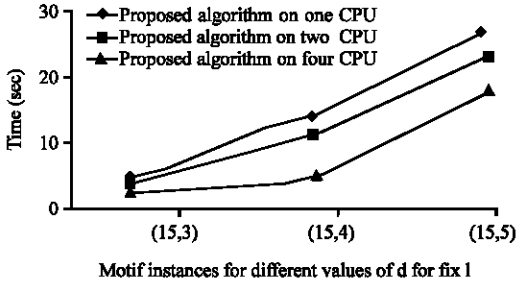

Fig. 4: Running time of our proposed algorithm on one, two and four CPU (1P, 2P, 4P) for l = 15 as a function of hamming distance of motifs

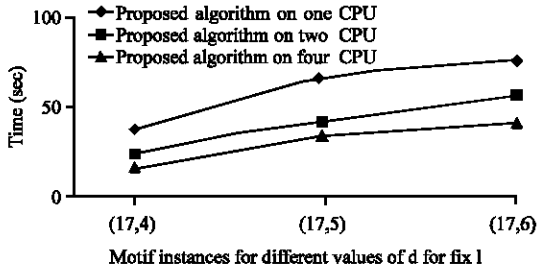

Fig. 5: Running time of our proposed algorithm on one, two and four CPU (1P, 2P, 4P) for l = 17 as a function of Hamming distance of motifs

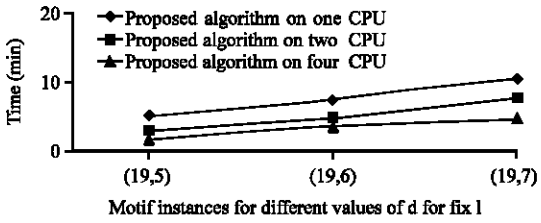

Fig. 6: Running time of our proposed algorithm on one, two and four CPU (1P, 2P, 4P) for l = 19 as a function of Hamming distance of motifs

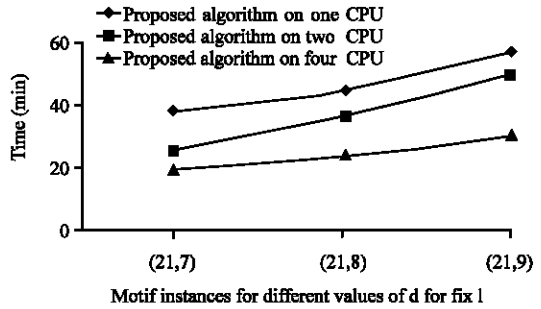

Fig. 7: Running time of our proposed algorithm on one, two and four CPU (1P, 2P, 4P) for l = 21 as a function of Hamming distance of motifs

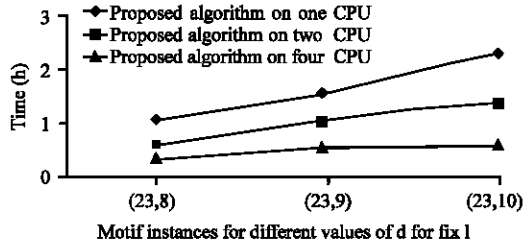

Fig. 8: Running time of our proposed algorithm on one, two and four CPU (1P, 2P, 4P) for l = 23 as a function of hamming distance of motifs

Hamming distance the curve indicates that the processing time has linear growth as the number of processors increases.

CONCLUSION

We present one efficient, simple, parallel bit vector approach for determining planted (l, d) motif. We initially introduced a fundamental way of getting the motif and then we do some modification which enhances the performance of the algorithm with less memory requirement. Although the bit vector has high memory requirement we minimize the use of memory by introducing the cumulative approach to higher sized l-mers. Use of bit vector simplifies the union and intersection of common neighborhoods. Here we eliminate the solving of any ILP instances and use of any lookup tables which minimise the memory consumption. The proposed parallel algorithm distributes the tasks of finding planted (l, d) motif from a set of sequences evenly among all the process in the cluster. Our experimental result justifies our claim that our proposed parallelization method on SMP cluster improves the running time over existing exact sequential and parallel algorithms. The experimental result on biological data demonstrates that our proposed algorithm competes in comparison to other exact algorithm.

REFERENCES

Bandyopadhyay, S., S. Sahni and S. Rajasekaran, 2014. PMS6: A fast algorithm for motif discovery. Intl. J. Bioinf. Res. Appl., 10: 369-383.

Buhler, J. and M. Tompa, 2002. Finding motifs using random projections. J. Comput. Biol., 9: 225-242.

Chin, F.Y. and H.C. Leung, 2005. Voting Algorithms for Discovering Long Motifs. In: Proceedings of the 3rd Asia-Pacific Bioinformatics Conference, Yi-Ping, P.C. and W. Limsoon (Eds.). Imperial College Press, London, UK., pp: 261-271.

Davila, J., S. Balla and S. Rajasekaran, 2006. Space and time efficient algorithms for planted motif search. Proceedings of the International Conference on Computational Science, May 28-31, 2006, Springer, Berlin, Germany, pp: 822-829.

Davila, J., S. Balla and S. Rajasekaran, 2007a. Fast and practical algorithms for planted (l, d) motif search. IEEE/ACM. Trans. Comput. Biol. Bioinf., 4: 544-552.

Davila, J., S. Balla and S. Rajasekaran, 2007b. Pampa: An improved branch and bound algorithm for planted (l, d) motif search. Master Thesis, University of Connecticut, Mansfield, Connecticut. https://pdfs.semanticscholar.org/d54a/d28de70daaf73e59f8334f182d7524187d7c.pdf.

Dinh, H., S. Rajasekaran and J. Davila, 2012. qPMS7: A fast algorithm for finding (l, d)-motifs in DNA and protein sequences. PloS One, Vol. 7,

Dinh, H., S. Rajasekaran and V.K. Kundeti, 2011. PMS5: An efficient exact algorithm for the (l, d)-motif finding problem. BMC. Bioinf., 12: 410-410.

Eskin, E. and P.A. Pevzner, 2002. Finding composite regulatory patterns in DNA sequences. Bioinf., 18: S354-S363.

Evans, P.A. and A.D. Smith, 2003. Toward optimal motif enumeration. Proceedings of the Workshop on Algorithms and Data Structures, July 30-August 1, 2003, Springer, Berlin, Germany, pp: 47-58.

Evans, P.A., A.D. Smith and H.T. Wareham, 2003. On the complexity of finding common approximate substrings. Theor. Comput. Sci., 306: 407-430.

Herts, G.Z. and G.D. Stormo, 1999. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. Bioinformatics, 15: 563-577.

Huang, C.W., W.S. Lee and S.Y. Hsieh, 2011. An improved heuristic algorithm for finding motif signals in DNA sequences. IEEE/ACM. Trans. Comput. Biol. Bioinf., 8: 959-975.

Keich, U. and P. Pevzner, 2002. Finding motifs in the twilight zone. Bioinf., 18: 1374-1381.

Lawrence, C.E., S.F. Altschul, M.S. Boguski, J.S. Liu, A.F. Newald and J.C. Wootton, 1993. Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignments. Science, 262: 208-214.

Nicolae, M. and S. Rajasekaran, 2014. Efficient sequential and parallel algorithms for planted motif search. BMC. Bioinf., 15: 1-34.

Nicolae, M. and S. Rajasekaran, 2015. qPMS9: An efficient algorithm for quorum planted motif search. MSc Thesis, University of Connecticut, Mansfield, Connecticut.

Pavesi, G., G. Mauri and G. Pesole, 2001. An algorithm for finding signals of unknown length in DNA sequences. Bioinf., 17: S207-S214.

Pevzner, P. and S.J. Sze, 2000. Combinatorial approaches to finding subtle signals in DNA sequences. Proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology, August 19-23, 2000, San Diego, CA., USA., pp: 268-278.

Pisanti, N., A.M. Carvalho, L. Marsan and M.F. Sagot, 2006. RISOTTO: Fast extraction of motifs with mismatches. Proceedings of the Latin American Symposium on Theoretical Informatics, March 20-24, 2006, Springer, Berlin, Germany, pp: 757-768.

Price, A., S. Ramabhadran and P.A. Pevzner, 2003. Finding subtle motifs by branching from sample strings. Bioinf., 19: ii149-ii155.

Rajasekaran, S. and H. Dinh, 2011. A speedup technique for (l, d)-motif finding algorithms. BMC. Res. Notes, 4: 54-54.

Rajasekaran, S., S. Balla and C.H. Huang, 2005. Exact algorithms for planted motif problems. J. Comput. Biol., 12: 1117-1128.

Rocke, E. and M. Tompa, 1998. An algorithm for finding novel gapped motif in DNA sequences. Proceedings of the 2nd Annual International Conference on Computational Molecular Biology, March 22-25, 1998, New York, USA., pp: 228-233.

Sagot, M.F., 1998. Spelling approximate repeated or common motifs using a suffix tree. Proceedings of the Latin American Symposium on Theoretical Informatics, April 20-24, 1998, Springer, Berlin, Germany, pp: 374-390.

Sun, H.Q., M.Y.H. Low, W.J. Hsu and J.C. Rajapakse, 2010a. RecMotif: A novel fast algorithm for weak motif discovery. BMC. Bioinf., 11L: S8-S8.

Sun, H.Q., M.Y.H. Low, W.J. Hsu and J.C. Rajapakse, 2010b. ListMotif: A time and memory efficient algorithm for weak motif discovery. Proceedings of the 2010 International Conference on Intelligent Systems and Knowledge Engineering (ISKE), November 15-16, 2010, IEEE, New York, USA., ISBN:978-1-4244-6793-8, pp: 254-260.

Sun, H.Q., M.Y.H. Low, W.J. Hsu, C.W. Tan and J.C. Rajapakse, 2011. Tree-structured algorithm for long weak motif discovery. Bioinf., 27: 2641-2647.

Sze, S.H. and X. Zhao, 2007. Improved pattern-driven algorithms for motif finding in DNA sequences. Proceedings of the Conference on Systems Biology and Regulatory Genomics, December 2-4, 2005, Springer, Berlin, Germany, pp: 198-211.

Sze, S.H., S. Lu and J. Chen, 2004. Integrating sample-driven and pattern-driven approaches in motif finding. Proceedings of the International Workshop on Algorithms in Bioinformatics, September 17-21, 2004, Springer, Berlin, Germany, pp: 438-449.

Yang, X. and J.C. Rajapakse, 2004. Graphical approach to weak motif recognition. Genome Inf., 15: 52-62.

Yu, Q., H. Huo, Y. Zhang a nd H. Guo, 2012. PairMotif: A new pattern-driven algorithm for planted (l, d) DNA motif search. PLoS One, Vol. 7.