Journal of Theoretical and Applied Information Technology
31st July 2016. Vol.89. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ENHANCED PLANTED(ℓ,D) MOTIF SEARCH PRUNE ALGORITHM FOR PARALLEL ENVIRONMENT

[1]SATARUPA MOHANTY, [2] BISWAJIT SAHOO

[1]Asstt Prof., School of Computer Engineering, KIIT University Bhubaneswar

[2]Assoc. Prof., School of Computer Engineering, KIIT University Bhubaneswar

E-mail: [1] Satarupafcs@kiit.ac.in, [2] bsahoofcs@kiit.ac.in

ABSTRACT

The Identification of inimitable patterns (motif) occurring in a set of biological sequences could give rise to new biological discoveries and has been studied considerably due to its paramount importance. In the field of bioinformatics, motif search is the vital problem for its application in the detection of transcriptional regulatory elements and transcription factor binding sites (TFBS) which are crucial for the knowledge of drug design, human disease, gene function etc. Many aspects of the motif search problem have been identified in the literature. One of them is the planted (ℓ, d)-motif search problem. In this paper, we propose a parallel extension of the existing PMS_{PRUNE} algorithm along with two additional features: those are neighbor generation on a demand basis and omitting the duplicate neighbor checking with the help of a bit vector implementation. The experimental result shows, the proposed multicore algorithm in C, handles the problem for larger d value with fix length ℓ with a speed up more than twice, without any additional requirement of space on a 2.4GHz PC with 4GB RAM.

Keywords: *PMS_{PRUNE}, Planted Motif Search, Symmetric Multiprocessor (SMP), Message Passing Interface (MPI), Bit Map Vector*

1. INTRODUCTION

In computational molecular biology and bioinformatics, the discovery of approximate patterns in DNA sequences has given rise to important solutions in the domain of many biological problems. For illustration, the recognition of patterns in protein sequences has ensured to be exceptionally helpful in domain recognition, locating the protease cleavage sites, identification of signal peptides, protein cooperation, resolution of protein degradation elements, identification of short functional motifs. Also, motifs are common sequence patterns in transcription factor binding sites that play significant roles in gene expression and regulation, understanding numerous human disease, gene function, drug design etc. As a result, in biological studies, the discovery of motifs plays an important and fundamental role. In literature, several versions of the motif search problem have been studied extensively. For the instance, they include Simple Motif Search (SMS), Edit-distance-based Motif Search (EMS), and Planted (ℓ, d) Motif Search. In this paper, our focus is on Planted (ℓ, d) Motif Search (PMS). PMS is defined as follows: Inputs to the PMS are n sequences of length m each, two positive integers' ℓ and d. The objective is to extract strings of length ℓ, such that any such string M is present in all n sequences with at most d mismatches. Formally, we can define the problem as follows:

Definition 1. Given a set of sequences $S := \{s_i\}_{i=1}^{n}$ over an alphabet ∑={A,T,C,G}, with $|s_i|$ = m and ℓ , d with $0 \le d < \ell < m$, the PMS involves identifying the (ℓ, d) motif x with $|x| = \ell$ 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 literature, PMS algorithms are categorized into two approaches: exact and approximation algorithms on the basis of heuristic search and exhaustive enumeration search respectively. Generally PMS approximation, algorithms are faster and more popular than exact algorithms but they are not guaranteed to give correct motif always. The probabilistic approach [1], [2], [3], [4] is utilized by the approximate algorithm, which depends on the Position-Specific Scoring Matrix (PSSM) representation [5], or a combinatorial

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

approach [6], [7], [8], [9], [10]. Extensively used analytical algorithms are the stochastic GibbsDNA algorithm [1], AlignACE [2], PhyloGibbs [3], BioProspector [4], TEIRESIAS [6], WINNOWER [7], Random Projection [8], MULTIPROFILER [9], Pattern Branching [10], CONSENSUS[11], MEME [12], MCEMDA[13] and Vine [14] . The WINNOWER algorithm, proposed by Pevzner and Sze[7] constructs a graph considering the ℓ-mers as nodes and there exist an edge between two ℓ-mers of different sequences if the mismatch among them is at most 2d. Then he maps PMS to the problem of finding a large clique, an NP-Hard problem. Random projections [8], by Buhler and Tompa, grouped ℓ-mers based on the similarity of the projections by considering k location from the entire ℓ-mers. The probability of getting the desired motif is more in the groups having a maximum number of ℓ-mers. Pattern Branching [10] algorithm uses the scoring method to assign a score to each neighbor of all ℓ-mers available in the sequence and then the best-scored neighbors are determined by the local search. The greedy CONSENSUS [11] employs statistical measures for the alignment of ℓ-mers and finds probable motifs from the alignment while GibbsDNA [1] uses Gibbs sampling. MEME (Multiple Expectation-Maximization Elicitation) [12] is one of the widely used approximation algorithms, where the technique of expectation minimization is used. A Monte Carlo algorithm, MCEMDA[13], initiate from a starting model and then it repeatedly executes the Monte Carlo simulation and parameter update till the convergence. Vine [14] a recent polynomial time Heuristic method based on WINNOWER[7].

Exact algorithms take exponential time to compute, but there is a guarantee of getting motifs. For the NP-hard nature of the exact algorithm, it is impractical to run on a very large instance and is therefore required to design the exact algorithm with small time and space overhead. Based on search space the exact algorithms built on two approaches: sample driven, test all $(m- \ell +1)^n$ possible combinations of ℓ-mers of different strings, generate the common neighbor and pattern driven, verify all 4^{ℓ} possible patterns to find the one that appear in all n sequences with minimum number of mutations. Some of the exact algorithms for solving PMS are SMILE [15], CENSUS [16], MITRA [17], PMS1 [18], PMS2 [18], Voting [19], and RISOTTO [20], three algorithms namely PMSi, PMSP, and PMSprune by Davila et al. [21], Pompa[22], PMS4[23], .Many of these algorithms use a tree data structure like a suffix tree or a mismatch tree or tries to steadily generate motifs one character at a time. CENSUS [16] constructs a trie out of all ℓ-mers from each of the input sequences. In the process of the trie generation, the nodes go on storing the number of mismatches from the motif, potentially pruning many branches of the trie. The algorithms SMILE[15] and RISOTTO[20] uses the suffix trees. Their theoretical space complexity is $O(n^2m)$ and time complexity is $O(n^2mN(\ell,d))$ where that $N(\ell,d) = \sum_{i=1}^{d}\binom{l}{d}(|\Sigma| - 1)^d$. Among this RISOTTO [20] performs efficiently and uses suffix tree. MITRA [17] considers the mismatch tree data structure. He divides the space of patterns into a number of sub-spaces those starts with a stated prefix and then employ pruning to each of those sub-spaces. PMS1, PMSi, PMS2 [18] are built on radix sort. They intersect efficiently the sorted d-neighborhood of all ℓ-mers in the input sequences using a different technique. Voting [19] uses the hash table to store all possible ℓ-mers and thus the space requirement is huge for large instances. Davila et al. [21] proposed a series of PMS algorithms those are fast as well as well as comparatively economical on space. PMSP [21] explores the d-neighborhood of the first input sequence's all ℓ-mers and then apply an extensive search with the rest input sequences to compute the particular ℓ-mers in the found d-neighborhood which are motifs. Practically it performs better than the PMS1 instead of its worse theoretical worst case time complexity. PMSprune [21] is an advancement over PMSP having time complexity as $O(nm^2N(\ell,d))$ with a space complexity of O(nm). This uses the branch and bound approach to exploring all d-neighborhood and uses dynamic programming to compute the distance among them. Pampa [22] added the concept of wildcard characters over PMSprune to compute approximate motif patterns and then finds the actual motifs by doing a thorough mapping within the computed approximate pattern. PMS4 [23], a generic speedup approach to improving the execution time of any exact algorithm. The algorithm PMS5 [24], PMS6[25] and qPMS7[26] computes all the common neighbors of the three ℓ-mers iteratively using an integer linear programming formulation. PairMotif[27] selects a candidate ℓ-mer from the input string and then alter the alphabets one by one. All the above-mentioned approaches use serial computation. A recently proposed bit based parallel approach [28], [29] implemented on multicore and GPU architecture are the bottleneck on memory requirement as the increase in the ℓ and d values.

The algorithms PMSprune[21], Pampa[22], qPMSPrunel[26], and qPMS7[27] are based on

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

search tree of depth d which was introduced by Davila in algorithm PMSprune[21]. In this paper, we work more efficaciously by minimizing the search tree space. We decide to work on PMSprune algorithm due to its advantages compared to other algorithms as discussed in [30]. We also develop a parallel version of PMSprune by using the SMP cluster for generation and process of many d-neighborhoods. The relative speed of it comes from the way of d-neighborhood generation and its intersection, and the relative space from the way of storing the huge ℓ-mers. This paper is organized as follows. The PMSprune[21] algorithm is described in section 2 and its limitations in section 3. In section 4 proposed algorithm has discussed, followed by the implementation detail in section 5. The discussion and comparisons on the results we present in section 6. Finally, the paper is ended with a conclusion in section 7.

2. EXISTING PMS$_{PRUNE}$ (A BRANCH AND BOUND ALGORITHM)

The PMSprune describes the solution of PMS in a geometric way through the concept of "distance" between a string and set of strings. We will use same notations and definitions as in [21] to state PMSprune.

input Let two strings x and y with $|x| = \ell$ and, $|y| = \ell$ the Hamming distance $d_H(x,y)$ can be defined as the number of places where the mismatch occur.

Definition 2. Given 2 strings with $|s| = m$ and $|x| = \ell$ respectively with ℓ <m. the notation $x \lhd_\ell s$ can be used to say x is a length ℓ substring of s or x is a ℓ-mer of s.

Definition 3. Given a string x with, $|x| = \ell$ the vicinity or neighborhood of x is denoted as $B_d(x) = \{y : |y| = \ell \text{ and } d_H(y,x) \le d\}$ where d_H is the Hamming distance.

Definition 4. Given two strings s and x of length m and ℓ respectively with ℓ <m, we denote by

$$\overline{d}_H(x,s) = \min_{x' \lhd_\ell s} d_H(x,x')$$

Definition 5. Given a string x with $|x| = \ell$ and a sequence of n number of strings $S := \{s_i\}_{i=1}^{n}$, with $|S_1| = m$, we denote by

$$\overline{d}_H(x,S) := \max_{i=1}^{n} d_H(x,s_i) = \max_{i=1}^{n} \min_{x' \lhd_\ell s_i} d_H(x,x')$$

Taking the advantage of these notation PMS$_{PRUNE}$ has described the definition 1 in a mathematical way as follows which is equivalent to say that x is the required (ℓ,d) motif.

$$\exists y \lhd_\ell s_1 : x \in B_d(y) \wedge \overline{d}_H(x,\{s_2,\ldots,s_n\}) \le d \qquad (1)$$

PMS$_{PRUNE}$ follow the equation 1. For every element of $B_d(y)$, while y is a ℓ-mer of s_1, it evaluates the function $\overline{d}_H(.,S)$. If it finds any x such as $\overline{d}_H(x,s) \le$ d, then it output x as the motif.

PMS$_{PRUNE}$[21] strategy: For each ℓ-mer y in s_1, it generate all possible neighbors of y and check whether that is a candidate motif or not. The idea of PMS$_{PRUNE}$ is given below.

a. It produces the neighbors for every ℓ-mer y of s_1 in a branch and bound strategy using the help of a tree T (y) of height d, where the root of the tree is ℓ-mer y and the children are the neighbors of it. If x is a neighbor of y, then x appears in T (y) at height h iff $d_H(x,y)=h$. Furthermore, if x' is a child of x in T (y) then $d_H(x,x')=1$. In figure 1 we have shown the example of such a tree.

b. The value of $d_H(x,S)$ and h are used to prune the descendants of x where x corresponds to a node in T (y).

c. Using the approach of construction of T (y) the neighborhood, it calculates the $\overline{d_H}(.,S)$ in an incremental way.

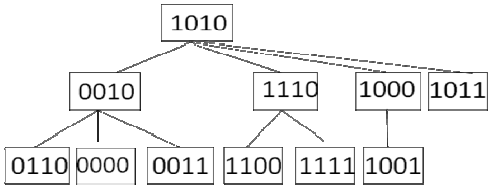

Figure 1. T (1010) with d=2 and ∑ = {0,1}

Algorithm PMS$_{PRUNE}$

For every $y \lhd_\ell S_1$

i) Traverse T (y) using a DFS strategy evaluating $\overline{d}_H(x,S)$, where x Є T (y).

ii) If($\overline{d}_H(x,S) \le$ d) ,then output x.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

iii) If($\bar{d}_H(x,S) - d > d - h$), then prune all the descendents.

iv) If ($\bar{d}_H(x,S)$ -d=d-h) , then consider only x' which has $\bar{d}_H(x',S) = \bar{d}_H(x,S)$ -1

v) If ($\bar{d}_H(x,S)$ -d=d-h-1), then consider only x' which has $\bar{d}_H(x',S) \leq \bar{d}_H(x,S)$

The $\bar{d}_H(x,S)$ can be calculated in a incremental way in a same procedure as [21].

3.LIMITATION

1. Here, for each ℓ-mer y of string S_1, it constructs all nodes of the tree T(y), then apply the pruning technique to discard the undesired nodes. However, the descendant nodes to the newly constructed node x are not required to be generated if $\bar{d}_H(x,S) \geq$ 2d-h.
2. It is assumed that the possibilities of occurring two ℓ-mers at a different position of string S_1 are dissimilar, but this may not hold for the large string. Again when two ℓ-mers are in close proximity, then duplicate neighbors can be generated. So, duplicate tree construction must be avoided. In overall there is no technique available which can restrict the generation of the neighbor for which already the motif search has taken place.

4.PROPOSED ALGORITHM

A series of algorithms Pampa[22], qPMSPrunel[26], and qPMS7[27] are based on the search tree mentioned in PMS_{PRUNE}[21]. In the extended PMS_{PRUNE} procedure, we add the following features in the neighbor generation technique of search tree as well as we paralyze that which improves the performance of the existing PMS_{PRUNE}[21].

1. Firstly in the tree neighbor node will not be generated unless it is neither a motif nor any of its children becomes a motif. This way of tree generation on demand basis we achieve with the help of recursive procedure.
2. Secondly we have avoided repeated generation of sub tree for same ℓ -mer of the string s_1 and for neighbors which are available earlier as the neighbor for some other ℓ-mer. In the implementation, we use bit vector. For every possible ℓ -mer or neighbor of length ℓ, there is a corresponding bit in the bit vector. The bit represents whether the ℓ-mer or neighbor has been already considered (1) or not (0) in the motif search process. For fixed length ℓ, increasing d, gives rise to tremendous increase in the number of neighbors but the same bit vector can able to hold those.
3. The efficiency is low in the serial implementation of the exact solution by single computer. Again most of the sequential algorithm contains lots of repeated, data-independent operations. This motivated us to propose a parallel way of implementing the existing exact planted (ℓ, d) motif search algorithm PMSprune[21] with some modification.

Each recursive call produces a derived ℓ -mer x' for the current ℓ -mer x. Here conditions must be checked before calling the recursive procedure for x' to generate the tree for node x'.

The relation between the current neighbor x and the derived neighbor x' are $\bar{d}_H(x,x') = 1$ and if S={$s_1,\ldots s_n$} then $\bar{d}_H(x,S) = \bar{d}_H(x',S) + \delta$ where $\delta = \{-1,0,1\}$ that is the value of $\bar{d}_H(.,S)$ does not differ too much between the neighbor and its derived one. Employing this property between current neighbor and its derived one, the following recursive call strategy is used.

Let y be a ℓ-mer of s_1 and x be a neighbor of y and let S={$s_1,\ldots s_n$} be a set of strings. Assume that $\bar{d}_H(x,S) = d + \Delta$ and x appears after h recursive calls, that is $\bar{d}_H(y,x) = h$, then the following condition or statements holds.

1. power If Δ > d-h, then none of the derived neighbors x' of x will be a motif hence generation of neighbors can be stopped.
2. If Δ = d - h, then only those derived neighbors x' having $\bar{d}_H(x',S) = \bar{d}_H(x,S) - 1$ may yield a motif thus be included in the search space of motif.
3. If Δ = d-h-1 then only those derived x' having $\bar{d}_H(x',S) \leq \bar{d}_H(x,S)$ may yield a motif and thus be included in search space of motif.

Here we have used two bit vectors M_1 and M_2 of size $2^{\ell\lceil\log|\Sigma|\rceil}$ each to store the neighbors of ℓ -mer y and motif respectively.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Algorithm 1. Modified $PMS_{PRUNE}()$

Input: ℓ, d, S= $\{S_i\}_{i=1}^{n}$

Output: Generate neighborhood of each ℓ-mer of S_1 by calling the function find_child().

```
1:  let Bit vector M1 and M2 initially empty.
2:  For each y ◁ℓ S[1] do
3:      For each pos=1 to ℓ do
4:          h=0
5:          find_child(pos, h, y)
```

Figure 2. Sequential modified PMSprune pseudocode

Algorithm 2. Find_child(pos, h, x)

Input: pos, current location of the ℓ-mer where the alphabet or symbol has to be changed h, current height of tree T(y) in the process of neighborhood generation, x, the ℓ-mer.

Output: Generate all neighborhood x' of ℓ-mer x for level h and check for the motif. If x' found to be the motif then save it in bit vector M_2 or otherwise depending on the value of $d_H(x',S)$ call function find_child_less() or find_child_less_equal().

```
1. Increment h
2:  For each ch=1 to |Σ| -1 do
3:      Find the neighbor of x as x' such that dH(x',x)=1, x[pos]≠x'[pos] and x' ∉ M1
4:      add x' to M1
5:      If(dH(x',S) ≤ d) then
6:          add x' to M2
7:          if(h!=d) then
8:              for pos=pos+1 to ℓ do
9:                  find_child(pos,h,x')
10:     else if(dH(x',S) –d=d-h) then
11:         if(d!=h)then
12:             for pos=pos+1 to ℓ do
13:                 find_child_less(pos,h,x')
14:     else if(dH(x',S)-d=d-h-1) then
15:         if(d!=h)then
16:             for pos=pos+1 to ℓ do
17:                 find_child_less_equal(pos,h,x')
```

Figure 3. Pseudocode to find neighbors of ℓ-mer x

Algorithm 3. Find_child_less(pos, h, x)

Input: pos, current location of the ℓ-mer where the alphabet or symbol has to be changed h, current height of tree T(y) in the process of neighborhood generation, x, the ℓ-mer.

Output: Generate level h neighborhood x' of ℓ-mer x where $d_H(x',S) < d_H(x.,S)$ and check for the motif. If x' found to be the motif then save it in bit vector M_2, otherwise depending on the value of $d_H(x',S)$ call function find_child_less()

```
1:  Increment h
2:  For each ch=1 to |Σ| -1 do
3:      Find the neighbor of x as x' such that dH(x',x)=1, x[pos]≠x'[pos] and x' ∉ M1
4:      add x' to M1
5:      if(dH(x',S) ≥ dH(x,S)) then
6:          return
7:      else if(dH(x',S) ≤ d) then
8:          add x' to M2
9:      else if(h!=d) then
10:         for pos=pos+1 to ℓ do
11:             find_child_less(pos,h,x')
```

Figure 4. Pseudocode to find neighbors x' of ℓ-mer x where $d_H(x',S) < d_H(x.,S)$

Algorithm 4. find_child_less_equal(pos, h, x)

Input: pos, current location of the ℓ-mer where the alphabet or symbol has to be changed h, current height of tree T(y) in the process of neighborhood generation. x, the ℓ-mer

Output: Generate level h neighborhood x' of ℓ-mer x where $d_H(x',S) <= d_H(x.,S)$ and check for the motif. If x' found to be the motif then save it in bit vector M_2, otherwise call function find_child_less_equal() .

```
1:  Increment h
2:  For each ch=1 to |Σ| -1 do
3:      Find the neighbor of x as x' such that dH(x',x)=1, x[pos]≠x'[pos] and x' ∉ M1
4:      add x' to M1
5:      if(dH(x',S) > dH(x,S)) then
6:          return
7:      else if(dH(x',S) ≤ d) then
8:          add x' to M2
9:      else if(h!=d) then
10:     for pos=pos+1 to ℓ do
```

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

11: find_child_less(pos,h,x'

Figure 5. Pseudocode to find neighbors x' of ℓ -mer x where $d_H(x',S) \leq d_H(x.,S)$

We have formulated the parallel version of the above sequential extended PMS_{PRUNE} where we have used the coarse grain and fine grain parallelism as shown below.

Algorithm 5. Parallel_ modified_PMS_{PRUNE}()

Input: ℓ, d, S= $\{S_i\}_{i=1}^{n}$

Output: In parallel it generates neighborhood of each ℓ-mer of S_1 and checks for motif

1. M_1= Ø and M_2= Ø
2. for each $y \triangleleft_{\ell} S[1]$ do parallel using process
3. for each pos=1 to ℓ do using thread
4. find_child(pos, 0, y);
5. M = ∪ M_2.

Figure 6. Pseudocode of parallel version of algorithm1

5. IMPLEMENTATION OF PROPOSED ALGORITHM

We have applied a Client-Server(C/S) mode to our LAN, where we have one Server and multiple clients. The Server coordinates and synchronizes all the clients. There is n number of sequences of length m each. The possible ℓ-mers of length ℓ in sequence s_1 are m- ℓ+1. Firstly we exploit the process-level (i.e course grain) parallelism by distributing the (m-ℓ+1)/p number of ℓ-mers of s_1, n-1 sequences and PMS tasks evenly to each client process (i.e the outer for loop of figure 6). The clients accept the individual nonoverlapping task with their domain and implement in a parallel way. Secondly we exploit the thread-level parallelism where the different thread finds the different set of neighbors of ℓ-mer y (i.e in the inner for loop of figure 6). Finally, the merging of the intermediate results produced on the clients takes place in the server side.

We have assumed the set of 'n' strings s:= $\{s_i\}_{i=1}^{n}$ is formed from the alphabet ∑= {A, C, G, T}. Each of these 4 distinct symbols can be represented by a binary string of length 2, that is A=00, G=01, T=10 and C=11. Using this phenomenon we consider every ℓ -mer as a sequence of the binary string. If $a_1, a_2, \ldots, a_{\ell}$ is an ℓ -mer, then this can be represented as $i_1, i_2, \ldots, i_{2\ell}$ binary string where each pair of binary digits corresponds to successive residues of ℓ -mer. Thus, a series of ℓ -residues can be replaced by an integer value in a natural way. For example when ℓ =16 and size of the integer is 32 bits (depending on the machine), each ℓ -mer can be represented as an integer. For each process, we have taken two-bit vectors of size half GB each for M_1 and M_2 to store the set of ℓ -mers which have gone through the motif search process and the set of found motif respectively. Each possible ℓ -mer in M_1 or motif in M_2 has corresponding bit position, which implies the presence (1) or absence (0) in the vectors. Bit vectors M_1 and M_2, initially contain all zeros. Whenever any ℓ -mer seems to be a motif or an ℓ -mer is found to be a motif, the bit position of the corresponding ℓ -mer is set to 1 in bit vector M_1 or M_2 respectively. Due to the use of bit vector here the repeated checking of same ℓ-mer can be avoided and thus the time of computation can be minimized.

5.1 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 distribute the task to the clients for the uniform computation. The clients return the produced results at the end.

Database sharing: All the bioinformatics information is stored in the main database on the server. The synchronization of the client is controlled by the server which is must 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 start to work. 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.

5.2 Client Designing

The clients simultaneously complete the task of getting DNA sequence into their database and computing Motif Finding from subsets of sequences under control of the server. For this we have another database on the clients which acts as a secondary database. In the client side, the download of the information starts from the server after receiving the tasks from it, and it stores those on the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

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 the clients with one End message to the server. The clients are configured with REDHAT Enterprise version-5 OS.

6. EXPERIMENTAL RESULT AND PERFORMANCE ANALYSIS

The algorithm that we proposed has been implemented in the C language using MPI library, Pthread library. The experiment is done in an environment where there is two nodes cluster with each of 2.4GHz Intel Pentium-IV processor having 2GB RAM running under Red Hat Linux. One Giga-bit/s 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 of transferring data or result between the server and the clients can be neglected as the distance of server and client is very short.

Synchronization delay (sd) :- to yield same the time of completion of the clients, the server divides the tasks basing on the number of files or sequences, by which the sizes of the subtasks are almost equal. The acceleration rate is mainly depends on synchronization delay due to the waiting of the server for the end message of its clients.

Data transfer delay (dt): -the clients download the sequences from the server and upload the result to the server. But due to voluminous data the Data transfer delay occur which is the main factor. The processing speed mainly depends on "Data transfer delay" and "Synchronization delay".

As mentioned previously, we have taken the number of sequences n=20 and length of each sequence as m=600 and experimented on a different value of ℓ and d. To show that our proposed new PMSprune run faster than the existing exact algorithms, we compare it against some of the previously fastest exact algorithms with different values of ℓ and d in Table1 below. In the parallel version of the proposed algorithm, the work to be done is distributed uniformly.

The first set of experiments is aimed at observing how lengths of the planted motif affect the performance of our parallel algorithm with respect to existing PMSprune sequential algorithm [21] as shown in Figure 7. It is observed in all cases that correct planted motifs are found using both proposed sequential and parallel algorithm successfully.

The next set of experiments investigates how the processing time is affected by varying Hamming distance for the different length of motifs. The Table 2 shows the behavior of new sequential and parallel PMSprune algorithm for various values of (ℓ,d).

Our parallel algorithm is executed by creating two number of processes on a two node cluster and four number of processes on four node cluster. This distributes one process to each node. Curves in figure 8 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 motif 15, 17, 19 as shown in figure 9, 10, and 11 respectively. Running time of the parallell agorithm on a two node cluster is compared with two nodes and four nodes, as well as with the new sequential PMSprune. With the increasing Hamming distance the curve indicates that the processing time has linear growth as the number of processors increase.

7. CONCLUSION

We have proposed and implemented parallel fast exact PMS algorithm by introducing a bit vector. The uses of bit vector minimize storage space requirement for a huge number of ℓ -mers generated during the tree construction by avoiding the redundant and repeated computation. This also produces motifs of higher Hamming distance (d). 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 system improves the running time over existing sequential algorithms [21] and [24]. We implement the proposed algorithm on two nodes and four nodes SMP cluster system with each of 2.4GHz Intel Pentium-IV having 4 GB RAM running under Red Hat Linux. The algorithm is also scalable i.e. by increasing the number of processes and number of sequences simultaneously maintains the speed up.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFRENCES:

[1] C. Lawrence, S. Altschul, M. Boguski, J. Liu, A. Neuwald, and J. Wootton, "Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment," Science, vol. 262, page 208-254, 1993.

[2] F.P. Roth, J.D. Hughes, P.W. Estep, and G.M. Church, "Finding DNA Regulatory Motifs within Unaligned Noncoding Sequences Clustered by Whole-Genome mRNA Quantization," Nature Biot echnology, vol. 56, page. 939-945, 1998.

[3] R. Siddharthan, E.D. Siggia, and E. van Nimwegen , "Phylogibbs: A Gibbs Sampler Incorporating Phylogenetic Information," PLoS Computational Biology, vol. 5, page 534- 556, 2005.

[4] X. Liu, J.S. Liu, and D.L. Brutlag, "BioProspector: Discovering Conserved DNA Motifs in Upstream Regulatory Regions of Co-Expressed Genes," Proc. Pacific Symp. Biocomputing, page 527, 2005.

[5] P.A. Evans, A. Smith, and H.T. Wareham, "On the Complexity of Finding Common Approximate Substrings", Theoretical Computer Sc., vol. 306, page 407-430, 2003.

[6] I. Rigoutsos and A. Floratos, "Combinatorial Pattern Discovery in Biological Sequences: The TEIRESIAS Algorithm", Bioinf., vol. 54, page. 56-57, 1998.

[7] P.A. Pevzner and S.H. Sze, "Combinatorial Approaches for Finding Subtle Signals in DNA Sequences", Intelligent Sys. For Molecular Biology, page. 269-278, 2000.

[8] J. Buhler and M. Tompa, "Finding Motifs Using Random Projections," J. Computational Biology, vol. 9, page 125-242, 2002.

[9] U. Keich and P.A. Pevzner, "Finding Motifs in the Twilight Zone," Bioinformatics, vol. 4, page 374-385, 2002.

[10] A. Price, S. Ramabhadran, and P. Pevzner, "Finding Subtle Motifs by Branching from Sample Strings," Proceeding. 2nd European Conf. Comput. Biology (ECCB '03), Bioinfo., suppl. ed., page 1-7, 2003.

[11] G.Z. Hertz and G.D. Stormo, "Identifying DNA and Protein Patterns with Statistically Significant Alignments of Multiple Sequences," Bioinf. Vol. 55, page 563-577, 1999.

[12] T. Bailey and C. Elka, "Fitting a Mixture Model by Expectation Maximization to Discover Motifs in Biopolymers", Proc. Second Int'l Conf. Intelligent Systems for Molecular Biology, page 28-36, 1994.

[13] Bi CP A monte carlo EM algorithm for De Novo motif discovery in bio molecular sequences. IEEE/ACM Trans. on Computational Biology and Bioinformatics-6: page 370–386, 2009.

[14] CW Huang, WS Lee, SY Hsieh An improved heuristic algorithm for finding motif signals in DNA sequences. IEEE/ACM Trans. on Computational Biology and Bioinformatics . page 959–975, 2011.

[15] L. Marsan and M.F. Sagot, "Extracting Structured Motifs Using a Suffix Tree-Algorithms and Application to Promoter Consensus Identification", Fourth Annual International Conference on Computational Molecular Biology (RECOMB), 2000.

[16] P. A. Evans and A. D. Smith , "Toward optimal motif Enumeration", Proc. Eighth Int'l Workshop Algorithm and Data structures (WADS 03), page 47-58, July/Aug. 2013.

[17] E. Eskin and P.A. Pevzner, "Finding Composite Regulatory Patterns in DNA Sequences," Bioinformatics, vol. 4, page 354-363, 2002.

[18] S. Rajasekaran, S. Balla, and C.-H. Huang, "Exact Algorithms for the Planted Motif Problem," J. Computational Biology,vol. 12, no. 8, page. 1117-1128, Oct. 2005.

[19] F.Y.L. Chin and H.C.M. Leung, "Voting Algorithms for Discovering Long Motifs", Proceeding 3rd Asia-Pacific Bioinfor. Conf., page 261-271, 2005.

[20] N. Pisanti, A.M. Carvalho, L. Marsan, and M.F. Sagot, "RISOTTO: Fast Extraction of Motifs with Mismatches", Proceeding 7th Latin Am. Theoretical Symp., page 757-768, 2006.

[21] J. Davila, S. Balla, and S. Rajasekaran, "Fast and Practical Algorithms for Planted(l,d) Motif Search,"IEEE/ACM Trans. Comp. Biology and Bioinf. vol 4,page 544-552, 2007.

[22] J. Davila, , S. Balla, and S. Rajasekaran, Pampa: An Improved Branch and Bound Algorithm for Planted (l, d) Motif Search, Tech Report, University of Connecticut, 2007.

[23] S. Rajasekaran, H. Dinh, A speedup technique for (l, d) motif finding algorithms, BMC Research Notes, 4:54,1-7, 2011.

[24] H. Dinh, , S. Rajasekaran, and V. Kundeti, PMS5: an efficient exact algorithm for the (l,d) motif finding problem, BMC Bioinformatics, 12:410, 2011.

[25] S. Bandyopadhyay, S. Sahni, and S Rajasekaran, "PMS6: A Fast Algorithm for Motif Discovery," Proc. IEEE Second Int'l

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Conf. Computational Advances in Bio and Medical Sciences (ICCABS '12),Feb. 2012 IEEE 2012:1-6.

[26] H. Dinh, S. Rajasekaran, and J. Davila, "qPMS7: A Fast Algorithm for Finding(l, d)-Motifs in DNA and Protein Sequences," PLoS One, vol. 7, no. 7, article e41425, July 2012.

[27] Q. Yu, H. Huo, Y. Zhang, and H. Guo, "PairMotif: A New Pattern-Driven Algorithm for Planted (l, d) DNA Motif Search," PLoS One, vol. 7(10), article e48442, Oct. 2012.

[28] N. S. Dasari, R. Desh, and M. Zubair, "An efficient multicore implementation of planted motif problem," in Proceedings of the International Conference on High Performance Computing and Simulation, pp. 9–15, 2010.

[29] N. S. Dasari, R. Desh, and M. Zubair, "Solving planted motif problem on GPU," in International Workshop on GPUs and Scientific Applications, 2010.

[30] D. Sharma, S Rajasekaran, H Dinh: An experimental comparison of PMSprune and other algorithms for motif search. CoRR abs, 1108.5217, 2011.

Journal of Theoretical and Applied Information Technology
31st July 2016. Vol.89. No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 1: Time Comparison Of Different Algorithms On Challenging Instances.

Instance / Motif / Algorithms	(11,3)	(13,4)	(15,5)	(16,3)	(17,6)	(19,7)	(21,8)	(23,9)
Proposed parallel PMSprune	3.07s	29.27s	4.23m	17.3m	25.85m	2.20h	8.98h	31h
Proposed Sequential PMSprune	4.6s	37.17s	6.92m	23m	47.03m	4.09h	10.39h	48h
PMS5	-	117s	4.8m		21.7m	1.7h	9.7h	54h
PMSprune	5s	53s	9m	-	69m	9.10h	-	
Pampa	4s	35s	5m	-	40m	4.45h	-	
PMSP	6.9s	152s	35m	-	12h	-	-	
Votting	-	104s	21.6m	-	-	-	-	
RISOTTO	-	772s	106.4m	-	-	-	-	

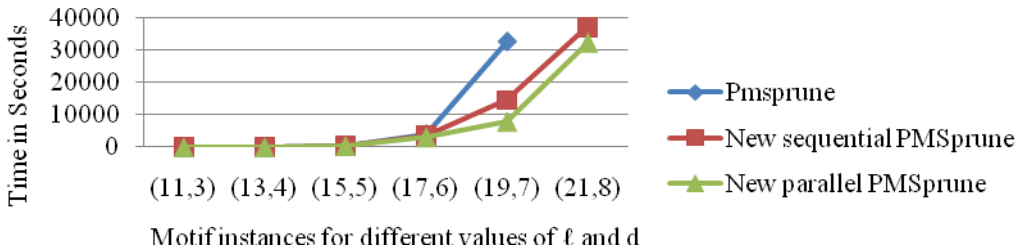

Figure 7: Running Time Of Proposed Sequential And Parallel Pmsprune Algorithm W.R.T The Existing Sequential Algorithm For N=20, And M= 600 As A Function Of (ℓ,D) Motifs For Different Values Of ℓ And D.

Table 2: Time Comparison Of New Pmsprune For Different (ℓ,D) Instances.

Length of the motif (ℓ)	**Hamming distance (d)**	**Processing time for New sequential PMSprune**	**Processing time for New parallel PMSprune**	
			Two CPU	**Four CPU**
15	5	6.92m	4.23m	3.01m
	4	4.34s	3.3s	2.05s
	3	0.182s	0.154s	0.114s
17	6	47.03m	25.85m	17.39m
	5	2.37m	1.96m	0.97m
	4	0.47s	0.43s	0.31s
19	7	4.09h	2.20h	1.18h
	6	132m	111m	42m
	5	1.023s	0.849s	0.611s

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

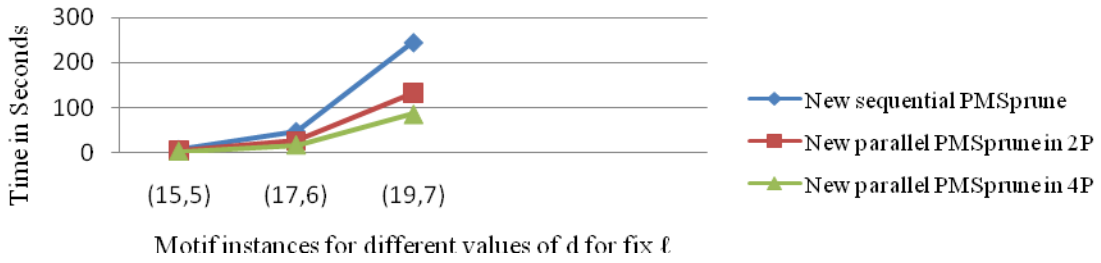

Figure 8: Running Time Of Our New Parallel Pmsprune Algorithm (2P,4P) And New Sequential Algorithm (SA) For N= 20, M= 600 As A Function Of (ℓ,D).

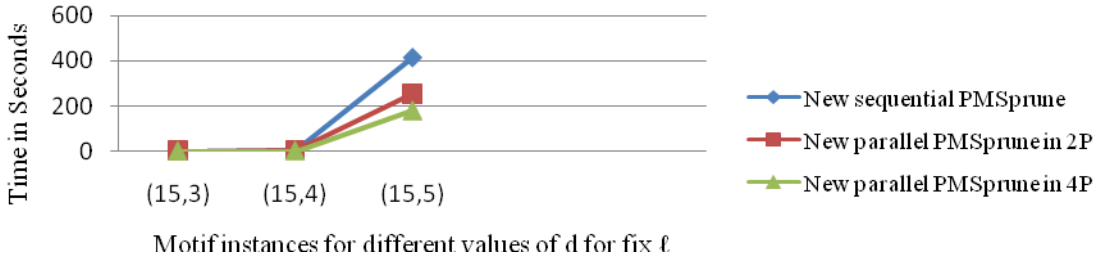

Figure 9: Running Time Of Our New Pmsprune And Parallel Pmsprune Algorithm (2P,4P) For N=20, M= 600 And ℓ = 15 As A Function Of Hamming Distance Of Motifs.

Figure 10: Running Time Of Our New Pmsprune And Parallel Pmsprune Algorithm (2P,4P) For N= 20, M= 600 And ℓ = 17 As A Function Of Hamming Distance Of Motifs.

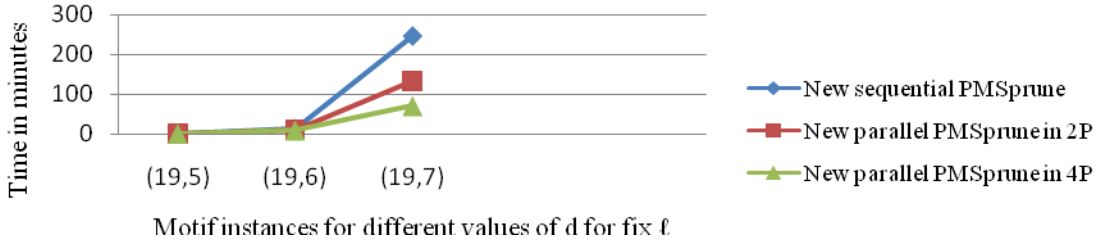

Figure 11: Running Time Of Our New Pmsprune And Parallel Pmsprune Algorithm (2P, 4P) For N= 20, M= 600 And ℓ = 19 As A Function Of Hamming Distance Of Motifs.