

## **Sequential PR\_FCP: A Big data Pattern Decomposing Algorithm To Identify Frequent Contiguous Patterns (FCP) From Large DNA Sequence Database Using Permutations With Repetition (PR)**

**S. Rajasekaran<sup>#1</sup>, L.Arockiam<sup>#2</sup>**

*<sup>#1</sup> R & D Centre, Bharathiar University, Coimbatore, India*

*<sup>#2</sup> Department of Computer Science, St. Joseph College, Trichy, India*

*<sup>#1</sup> [sraja2911@gmail.com](mailto:sraja2911@gmail.com)*

*<sup>#2</sup> [larockiam@yahoo.co.in](mailto:larockiam@yahoo.co.in)*

### **Abstract**

One of the challenges in this Big Data era is to extract patterns from the large volumes of data. DNA sequences are huge in volume and size. Finding patterns from these data is a tedious process. Frequent Contiguous Patterns (FCP) are the patterns which are repeatedly occurring in the database. A major challenge in mining frequent contiguous patterns from a large DNA sequence database (SDB) is the generation of huge number of FCPs satisfying the minimum support (min\_sup) threshold, especially when it is set to low. The task of discovering all these FCPs in large DNA databases is quite challenging as the search space is extremely large. With millions of sequences in the DNA SDB the problem of I/O minimization becomes very critical. Most current algorithms are iterative in nature which requires repeated scanning of full SDB which is a very expensive process. This huge searching space causes delay in transferring the required data between storage and the buffer/memory. The proposed algorithm Sequential PR\_FCP reduces the search space by dividing the main DB into smaller sub DBs based on possible patterns existing in the main DB. The proposed algorithm aimed to find interesting \_FCP of length\_4 by applying Partitioning the SDB, Prune the search space using Vertical Data format methodologies. Identifying these patterns from the DNA sequences helps to understand the function and structures of DNA. It also helps to identify Motif, Regulatory regions, ORF and internal repeats.

**Keyword:** Frequent Contiguous Pattern; DNA sequences; Motif/Regulatory regions; Big Data; Cache optimization; Permutations with Repetition (PR)

## Introduction

In this Big Data era, the data are huge in size and volume. It is used to say that more data beats the best algorithms. The main problem in handling the Big Data is to store and retrieve the data in an efficient manner with less processing time over data. It is very critical to optimize the usage of cache memory while processing the big data.

A sequence database (SDB) is the collection of data sequences. These data sequences contain a fixed number of data items occurring repeatedly. Generally, in life-science field, all the DNA and Protein sequences consist of long linear chain of chemical components. DNA sequences contain four nucleotide characters A, G, T and C namely Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). A genome is the complete set of genes of an organism. DNA SDBs are huge in size and with large volume contain millions of sequences with billions of repeated patterns. SDB applications require to identify frequently occurring common patterns, referred as Frequent Contiguous Patterns (FCP) existing in the sequences of the SDB.

There exist more similarities in nucleotides and protein sequences of species. Sequence alignment, a process which lines up these bio-sequences, helps to achieve highest level of similarities between the sequences. Sequence alignment primarily identifies same or similar sequences with long conserved subsequences as between two or more bio-sequences. Those two sequences which share the common ancestor are referred as homologous. The degree of similarity aims to find the homology possibility between two sequences. Similarity in bio-sequence primarily aims to determine the relative position of multiple species in phylogenetic tree, an evolution tree. In DNA SDB applications, FCP aims to identify regulatory regions, which are part of DNA sequences and are responsible for regulating the genes. And FCPs in DNA sequences are very useful to understand the function and structure of DNA and helps to find Motif, Regulatory regions, ORF and internal repeats etc.

There are three different approaches are widely used in FCP findings namely, sequential patterns mining, structured patterns mining and scalable patterns mining. The mining of frequent subsequences in a sequence data set is referred as sequential patterns mining. The mining on structured data sets like trees, graphs, sequences, combination of above structures are referred as structured pattern mining.

Sequential pattern mining searches for frequent sub-sequences in a sequence database (SDB), where a sequence record contains {seq\_id, sequence}. "Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items, and given a user-specified min support threshold, sequential pattern mining is to find all of the frequent sub-sequences, i.e., the sub-sequences whose occurrence frequency in the set of sequences is no less than min support"[1].

The task of discovering all frequent sequences in large databases is quite challenging. The search space is extremely large. For example, with  $m$  sequences there are  $O(m^k)$  potentially frequent sequences of length  $k$ . With millions of sequences in the SDB the problem of I/O minimization becomes very critical. Most current algorithms are iterative in nature which requires repeated scanning of full SDB is a very expensive process.

A major challenge in mining frequent contiguous patterns from a large DNA SDB often generates a huge number of FCPs satisfying the minimum support (min\_sup)

threshold, especially when it is set low. This is because if a pattern is frequent, each of its sub-patterns is frequent as well. A long pattern will contain a combinatorial number of shorter, frequent sub-patterns. For example, a frequent pattern of length 100, such as  $\{a_1, a_2, \dots, a_{100}\}$ , contains  $\binom{100}{1}=100$  frequent length\_1 pattern:  $\{a_1, a_2, \dots, a_{100}\}$ , contains  $\binom{100}{2}$  frequent length\_2 patterns:  $(a_1, a_2), (a_1, a_3), \dots, (a_{99}, a_{100})$ , and so on. The total number of frequent contiguous patterns present in the SDB is as given below:

$$\binom{100}{1} + \binom{100}{2} + \dots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}$$

These numbers of computations require high computational time and large storage space. A set of transactions in a horizontal format (that is,  $\{\text{seq\_id: sequences}\}$ ), where seq\_id is a sequence-id and sequence. The same data can also be presented as  $\{\text{sequences: seq\_id}\}$ , where sequences represents possible FCPs and seq\_id is the set of seq\_ids containing the respective possible FCP. This format is known as vertical data format.

The vertical data format improves the process of mining FCPs. In this process first we transform the horizontally formatted data to the vertical data format by scanning the SDB once. The support count of a possible FCP is simply the length of the seq\_id set. Starting with  $k = 1$ , the frequent  $k$ -patterns can be used to construct the candidate  $(k+1)$ -patterns. The computation is done by intersection of the seq\_idsets of the frequent  $k$ -patterns to compute the seq\_id sets of the corresponding  $(k+1)$ -patterns. This process repeats, with  $k$  incremented by 1 each time, until no frequent patterns or no candidate patterns can be found. Another merit of this method is that there is no need to scan the database repeatedly to find the support of  $(k+1)$  patterns (for  $k \geq 1$ ).

All DNA sequences comprised of four basic building blocks called nucleotides Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). As, biological sequences normally consist of millions of amino acid bases, the efficient and accurate discovery of frequent contiguous sequences with a minimal time, poses a great challenge to pattern mining algorithms.

The rest of this paper is organized as follows. Section 2 describes problem definition and discusses the related work to find FCP. Section 3 proposes a new efficient algorithm for FCP called Sequential PR\_FCP Section 4 analyses the proposed algorithm Sequential PR\_FCP and its implementation details. Section 5 concludes with future research direction..

## Problem Definition and Related Work

### A. Problem Definition

Let us assume  $I = (i_1, i_2, \dots, i_k)$  is an item set of size  $k$  and  $B = \{(s_1, a_1), (s_2, a_2), \dots, (s_n, a_n)\}$  is a Sequential Database (SDB) of size  $n$ , where  $s_i$ , are sequence ids and  $a_i$ , are the database sequences consists of items in  $I$  and each of its length is  $m$ , where  $1 \leq i \leq n$ .

n.FCP identification is the process of finding repeatedly occurring contiguous patterns of length n in sequences  $a_i$ ,  $1 \leq i \leq n$ , over the SDB, namely B.

For instance, let the Item set  $I = (A, G, T, C)$   
 and SDB,  $B(s_i, a_i) =$   
 $\{ (1, \text{GTG } \underline{\text{ATCG}} \text{ACT } \underline{\text{ATGC}}),$   
 $(2, \underline{\text{ATCG}} \text{CTTC } \underline{\text{ATCG}})$   
 $(3, \text{CTGAAGT} \underline{\text{ATCG}})$   
 $(4, \text{ATTGTCGTG})$   
 $(5, \text{GTGGCATTG}) \}, 1 \leq i \leq 5$

**Figure 1:** FCP Example for Biological SDB

Fig.1 is a sample DNA SDB, which contains five DNA sequences. ATCG is a FCP for the above sample SDB.

### B. Related work

Apriori[2] is widely used algorithm to find FCP. The property of Apriori is “All Non-empty subsets of a frequent item set must also be frequent”. i.e. If a subset does not poses a minimum support threshold then its super set also failed to be frequent. This level wise algorithm generates 1-item dataset  $L_1$ , 2-item dataset  $L_2$ .... K-item dataset  $L_k$ . All these datasets are obtained by repeatedly scanning of database by searching frequent data items. Apriori property helps to reduce the searching space. The disadvantage of this method is repeated scanning of database and considering all possible frequent sets which involve huge computation and memory.

SP-Index (Segment to Pattern Index) algorithm [3] contains two phases namely Segment phase and Pattern index phase. In Segment phase, it finds the entire base frequent pattern list. In Pattern index phase, SP-Index tree connects the base pattern with its consecutive short regions [4]&[5]. The formation of SP-Index tree for all existing base pattern requires high computation and memory.

Surprising contiguous pattern mining algorithm [6] constructs the Index based spanning tree for the patterns and compares the leaf nodes with new probability based support called “minimum Information gain threshold” and “minimum confidence threshold” [7]. The leaf nodes of spanning tree contain positions, sequenceid of the patterns. Using these information the required FCP can be found. [8-10].

Location based FCP algorithm [11] is based on Apriori technique. This uses a new and intelligent way for sorting and joining of the required base pattern table which can quickly produce new patterns than the traditional methods. The position based Fast Contiguous FCP algorithm [12] constructs spanning tree for patterns with possible Information and perform joining operations to find next level patterns with the help of hash table and binary search.

To improve time and space complexity, all the above algorithms uses a number of intermediate tables, Spanning Tree, hashing and joining techniques.

### Sequential PR\_FCP: Proposed Algorithm To Identify FCP

The FCP identification requires huge searching space for all possible base patterns existing in the SDB. This huge searching space causes delay in transferring the required data between storage and the buffer/memory. The efficient use of the buffer reduces the time and space complexity for finding FCPs. The best way to reduce the buffer size/searching space is transferring smaller chunk of data to the buffer.

The proposed algorithm Sequential PR\_FCP reduces the search space by dividing the main DB into smaller sub DBs based on possible patterns existing in the main DB. Thus it optimizes the cache memory usage by transferring smaller chunk of data sequences into cache than large volume of complete main DB.

In this section, we propose an algorithm which is based on Decomposing of SDB into vertical data format with respect to all possible number of permutations with repetition (PR) patterns. The proposed Sequential PR\_FCP aimed to find interesting \_FCP of length\_4 by applying following methodologies:

- Partitioning the SDB – Decomposing SDB with respect to PR
- Prue the search space – Vertical Data format

DNA sequence database  $D$  is a set of tuples  $(seq\_id, S)$  where  $seq\_id$  is a sequence identifier and  $S$  is the corresponding sequence formed with characters (A,T,C,G) only. As per Enumerative Combinatorics the PR [13] is referred as follows “Ordered arrangements of the elements of a set  $S$  of length  $n$  where repetition is allowed are called  $n$ -tuples, but have sometimes been referred to as permutations with repetition although they are not permutations in general”. If the set  $S$  has  $k$  elements, the number of  $n$ -tuples over  $S$  is  $k^n$ . There is no restriction on how often an element can appear in an  $n$ -tuple.

The DNA sequences consist only of four characters A, T, C and G. So the base patterns of length-2 should be from any one of the following possible 16 ( $2^4$ ) PR patterns namely

{AA, AT, AC, AG,  
TA, TT, TC, TG,  
CA, CT, CC, CG,  
GA, GT, GC, GG}.

The proposed algorithm Sequential PR\_FCP decomposes the large DNA main DB into 16 sub DBs namely {tblAA, tblAT, tblAC, tblAG..... tblGG} and stores the possible PR patterns into vertical data format. The proposed algorithm consists of two major methods called Decompose DB() and Interesting\_FCP4(), along with few accessory methods. The method Decompose DB() decomposes each sequence of the main DB into consecutive sub-sequences of length-4. And stores these sub-sequences in vertical data format. These length\_4 patterns map into respective sub DBs which is generated by the method called subDBcreation().

The method sub DB creation() maps the length-4 sequence patterns into its corresponding 16 subDBs, based on their first two characters. After executing Decompose DB(), the 16 sub DBs will have all possible length-4 sequence patterns of the main DB with their respective seq\_id, and their starting position in the sequence.

**Table 1:** Example Sequences

Seq_ID	Sequence
10	ATCGCTTCATCG

For instance the sequence in Table 1, will be sliced out by length-4 consecutive patterns and stored in their respective subDBs given by subDBcreation() method. Table 2 shows sample results for the sample sequence in Table 1.

**Table 2:** sub DBs generated by sub DB creation()

tblAT contains	{(10, ATCG,1), (10,ATCG,9)}
tblAC contains	{(10, TCGC,2), (10, TCAT,7)}

The next method Interesting\_FCP4() identifies the respective sub DB from the 16 sub DBs based on the first two characters of the given input. For instance, if the required FCP is 'ATCG' then Interesting\_FCP4 () method selects the sub DB 'tblAT'. Then it scans the selected 'tblAT' and if matching occurs it stores the result into the output table namely reqfcp4, with its respective seq\_id, starting position of the length-4 pattern. To get 'ATCG', it is enough to compare sub DB 'tblAT' only.

### PSEUDO Code

The pseudo code for the proposed algorithm Sequential PR\_FCP is as given below:

```

Algorithm SequentialPR_FCP()
{
//Let S1,S2...Sn are the sequence_ids.
//Let a1,a2...an are the sequences
//n is the size of the SDB
//m is the sequences length
//(p1,p2...pk) is the required FCP
//creation of 16 PR_patternsubtables
for i= 1 to n
{
read s[i],a[i];
n1=s[i];
// Get the consecutive characters (x,y) of ai
for j=1 to m
{
x=substring(a[i], j, j+1);
start_position = j;
PR_pattern = substring(a[i], j, j+3);
y = SubDBcreation(x);
DecomposeDB(y, PR_pattern, n1, start_position);
} // End j for loop
}
}

```

```

} // End I for loop
//Get the first two characters (x,y) of required_FCP
x=substring(required_FCP,1,2);
//Get the required_FCP's PR_Patternsubtable x
//m1 is the size of PR_patternsubtable x
//Search the required_FCP in the Possible subset
For k=1 to m1
{
if(strcmp(PR_pattern, required_FCP) == 0)
print(seq_id,PR_pattern,position);
} // end for k
} //end SequentialPR_FCP()
Procedure DecomposeDB(string tbl_name, string PR_pattern, intseq_id,
position_id)
{ Insert into table "tbl_name" values "PR_pattern, seq_id, position_id";}
String Function reindex_creation(string s1)
{
String x, y;
String x = s1;
Switch(x)
{
Case "AA" : {y="tblAA"; return y; break};
Case "AT" : {y="tblAT"; return y; break};
Case "AC" : {y="tblAC"; return y; break};
Case "AG" : {y="tblAG"; return y; break};
Case "CA" : {y="tblCA"; return y; break};
Case "CT" : {y="tblCT"; return y; break};
Case "CC" : {y="tblCC"; return y; break};
Case "CG" : {y="tblCG"; return y; break};
Case "TA" : {y="tblTA"; return y; break};
Case "TT" : {y="tblTT"; return y; break};
Case "TC" : {y="tblTC"; return y; break};
Case "TG" : {y="tblTG"; return y; break};
Case "GA" : {y="tblGA"; return y; break};
Case "GT" : {y="tblGT"; return y; break};
Case "GC" : {y="tblGC"; return y; break};
Case "GG" : {y="tblGG"; return y; break};
}
break;
} //end subDBcreation method

```

**Figure 2:** The pseudo code of Sequential PR\_FCP

Thus the proposed algorithm Sequential PR\_FCP narrows down the search space to a small subDB than the large and complete main DB. Hence it optimizes the cache

usage by transferring smaller data and reduces the runtime, by performing linear comparison of the data in the sub DB.

## Time and Space Analysis of Sequential PR\_FCP

### A. Time and Space Analysis of Sequential PR\_FCP

Let, 'm' is the number of sequences in main DB, 'l' is the length of each sequence in main DB and 'n' is the pattern length. And  $(l - n + 1)$  is the number of slicing in each sequence, then the number of comparisons needed to find required FCP of length-n in main DB is derived from the below given Eq.1.

$$\begin{aligned} T(m, l) &= m * l * (l - n + 1), \text{ where } l > n \text{ -----} & \text{Eq.1} \\ &= m * l * (l - n + 1) \\ &= O(m * l^2) \end{aligned}$$

The proposed algorithm divides 'm' sequences of main DB into 16 sub DBs of length-4. These sub DBs contain approximately '4m' times of length-4 sequences. Hence, the Eq.1 is reduced by replacing 'm' as '4m', 'l' as 'n' and slicing factor  $(l - n + 1)$  as  $(n - n + 1)$  which is 1 (as sub DB contains sequence of length-4). Thus Eq. 1 is reduced as

$$\begin{aligned} T(4m, n) &= 4m * n * 1 \text{ -----} & \text{Eq.2} \\ &= O(4m * n) \end{aligned}$$

The ratio of number of comparisons between main DB and sub DB by Eq.1 and Eq.2

$$\begin{aligned} T(m, l) / T(4m, n) &= ((l * (l - n + 1)) / (4 * n)) \text{ -----} & \text{Eq.3} \\ &= O(l^2 / n) \end{aligned}$$

Thus proposed Sequential PR\_FCP algorithm reduces  $O(l^2 / n)$  times of time complexity than the traditional sequential comparison to find the FCP in the main DB.

The main DB contains 'm' sequences of length 'l'. The cache memory needed for main DB is 'm \* l'. The sub DB contains approximately '4m' sequences of length-4. Sub DB requires  $16m = \approx (n^2 m)$  cache memory. Hence the space complexity of proposed Sequential PR\_FCP is given in Eq.4.

$$S(m, n) = O(l / n^2), \text{ where } l > n^2 \text{ -----} \quad \text{Eq.4}$$

The proposed algorithm Sequential PR\_FCP reduces  $(1 / n^2)$  cache memory usage of sub DB than the cache memory usage main DB.

### B. Implementation Details

Human genome DNA sequences (Homo sapiens GRCh38 DNA Chromosome) are downloaded from the NCBI website (<http://www.ncbi.nlm.nih.gov/nucleotide/>) for the analysis of the proposed Sequential PR\_FCP algorithm. The selected human genome DNA database contains 1,34,497 sequences with each sequence of length 70. All the programs are written in Java 1.7.0\_65 and the My SQL 5.5.38-0 is used as the

database. Ubuntu 12.04.01 is the base OS with Intel Core i7 CPU @ 2.00 GHz x 4 with 8 GB main memory and 25 GB hard disk.

**Table 3:** Database Details

DB Name	Seq #	Seq Len	# of Slicing expected for Sub DB (by Eq.1)	Buffer Usage (MB)	Storage (MB)
Main DB	1,34,947	70	63,29,01,430	746.29	14
tblAA	9,09,847	4	36,40,356	34.43	37
tblAT	7,25,000	4	29,00,916	91.74	30
tblAC	4,62,416	4	18,51,796	125.58	19
tblAG	6,12,165	4	24,50,616	113.21	25
tblTA	6,08,895	4	24,37,976	146.90	25
tblTT	9,02,598	4	36,11,916	133.71	37
tblTC	5,31,220	4	21,26,716	173.21	22
tblTG	6,61,921	4	26,49,696	121.01	27
tblCA	6,59,046	4	26,37,056	150.21	27
tblCT	6,14,065	4	24,58,516	151.42	25
tblCC	4,38,709	4	17,56,996	146.23	18
tblCG	78,707	4	3,16,012	137.05	4
tblGA	5,31,560	4	21,28,296	125.95	22
tblGT	4,62,347	4	18,50,216	127.70	19
tblGC	3,58,669	4	14,34,676	163.51	15
tblGG	4,39,727	4	17,60,156	212.47	18

The implementation of Sequential PR\_FCP is summarized in Table.3 which provides the size, number of comparisons, buffer/cache usage and the disk storage of each 16 sub DBs and main DB. The formation of 16 sub DBs is the one time effort only. After the sub DBs formation the FCP will be found by a linear search over each respective sub DB. Hence the runtime, space complexity for sub DB FCP is number of records in the respective sub DB, which are linear.

The average number of sequences ( $4m$ ) of 16 sub DBs' is 5,62,686 which is 4 times of number of sequences ( $m$ ) of main DB (1,34,947). The average number of comparisons of 16 sub DBs' is 22,50,745, which is approximately 293 times (by substituting  $l = 70$ ,  $n=4$  in Eq. 3) less than the number of comparisons of main DB (63,29,01,430). Thus the time complexity is reduced to  $O(l^2 / n)$  times than the number of comparisons in main DB. The average buffer/cache memory usage by 16 sub DBs is 134.65 MB, which is 4.19696 times of main DB cache usage (746.29 MB). This can be verified by substituting  $l=70$ ,  $n = 4$  in Eq.4.

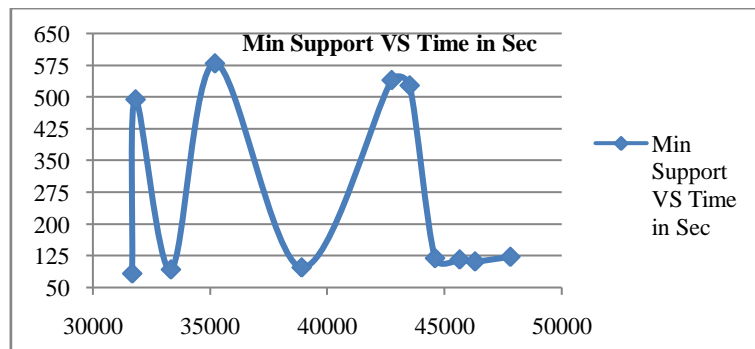
### C. Results Analysis and Discussion

A DNA sequence is sequence of only four item set namely A,G,T and C. Hence, the short pattern of Length-4 repeatedly occurs several times (even within the same sequence) which makes the possibility of obtaining minimum support to the multiples of 1000s. But in the market-basket analysis, the sequences consist of several item set, which makes the possibility of obtaining minimum support in smaller amount. Thus DNA sequence contains larger amount of minimum support, whereas market-basket sequence contains smaller amount of minimum support for small pattern (length-4).

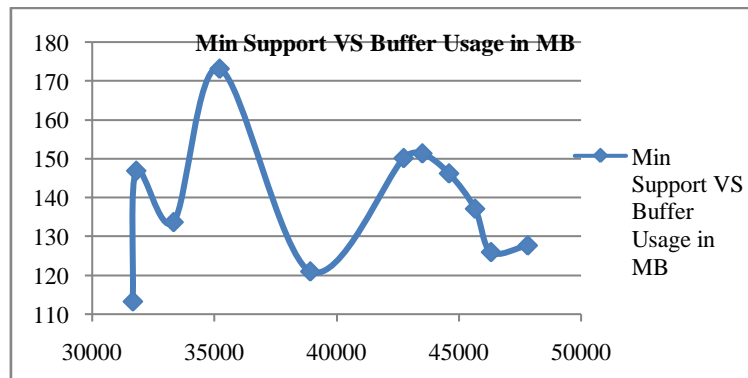
**Table 4:** Runtime Results of Sample Pattern

Sub DB	# Records	Pattern	Linear Comparison	Min Sup	Buffer (in MB)	Time (in Sec)
tblGT	4,62,347	GTAT	4,62,347	31666	113	84
tblAT	7,25,000	ATAC	7,25,000	31810	147	494
tblTA	6,08,895	TAAG	6,08,895	33336	134	92
tblTT	9,02,598	TTAG	9,02,598	35202	173	580
tblAC	4,62,416	ACAC	4,62,416	38910	121	98
tblCA	6,59,046	CATG	6,59,046	42733	150	540
tblTG	6,61,921	TGAT	6,61,921	43499	151	528
tblAG	6,12,165	AGTT	6,12,165	44585	146	120
tblTC	5,31,220	TCAG	5,31,220	45648	137	117
tblCC	4,38,709	CCAG	4,38,709	46302	126	113
tblGG	4,39,727	GGAA	4,39,727	47802	128	123

Table.4 provides runtime details of 11 sample patterns. Sequential PR\_FCP requires minimum time of 84 seconds and 113 MB buffer/cache usage to fetch minimum support of 31,666 records. Fig.3 provides the runtime of Sequential PR\_FCP for the sample patterns in Table.4.



**Figure 3:** Runtime of Sequential PR\_FCP for the sample pattern



**Figure 4:** Buffer usage by Sequential PR\_FCP for the sample pattern

The proposed algorithm Sequential PR\_FCP requires a minimum of 113 MB and maximum of 171 MB buffer/cache usage for the minimum support from 31,666 to 47,802. Fig 4.3 provides the buffer usage details for the sample pattern in Table.4.

### Conclusion and Future Research Direction

The proposed algorithm Sequential PR\_FCP reduces the search space by dividing the main DB into smaller sub DBs based on possible patterns existing in the main DB. Thus it optimizes the cache memory usage by transferring smaller chunk of data sequences into cache than large volume of complete main DB. This algorithm is based on Decomposing of SDB into vertical data format with respect to all possible number of permutations with repetition (PR) patterns.

The proposed algorithm consists of two major methods called Decompose DB() and Interesting\_FCP4(), along with few accessory methods. The method Decompose DB() decomposes each sequence of the main DB into consecutive sub-sequences of length-4. And stores these sub-sequences in vertical data format. These length\_4 patterns map into respective sub DBs which is generated by the method called sub DB creation(), which maps the length-4 sequence patterns into its corresponding 16 sub DBs, based on their first two characters. The next method Interesting\_FCP4() identifies the respective sub DB from the 16 sub DBs based on the first two characters of the given input. Thus the proposed algorithm Sequential PR\_FCP narrows down the search space to a small sub DB than the large and complete main DB. Hence it optimizes the cache usage by transferring smaller data and reduces the runtime, by performing linear comparison of the data in the sub DB.

In future, this algorithm can be extended for length-n, by comparing consecutive positions of the DNA sequences. This can be done by combining multiple subDBs in parallel by using these 16 subDBs.

## References

- [1] Agrawal R, Srikant R. 1994. Fast Algorithms for Mining Association Rules. Proceedings of the 20th VLDB conference, Santiago
- [2] Srikant R., Agrawal R. 1996. Mining sequential patterns: Generalizations and performance improvements. 5th International Conference on Extending Database Technology, Avignon, France
- [3] Pan J, Wang P, Wang W, Shi B and Yang G. 2005. Efficient algorithms for mining maximal frequent concatenate sequences in biological datasets. In Proceedings of the Fifth International Conference on Computer And Information Technology (CTT), PP: 98-104.
- [4] Yang, J., Wang, W., Yu, P. S. & Han, J., 2002. Mining long sequential patterns in a noisy environment. SIGMOD.
- [5] Brazma A., Jonassen I., Eidhammer I. and Gilbert D. 1995. Approaches to the automatic discovery of patterns in biosequences. Technical report, Department of Informatics, University of Bergen, Norway, 1995.
- [6] Rashid, M., Karim, M. R., Jeong, B. & Choi, H., 2012. Efficient Mining of Interesting Patterns in Large Biological Sequences. Genomics & Informatics, 10(1), pp. 44-50.
- [7] Blahut, R., 1987. Principles and Practice of Information Theory, Boston, MA: Addison-Weley Longman Publishing Co., Inc
- [8] Kang, T. H., Yoo, J. S. & Kim, H. Y., 2008. Mining frequent contiguous sequence patterns in biological sequences. Athens, 7th IEEE International Conference on Bioinformatics and Bioengineering (BIBE'08).
- [9] Zerín, S. F., Ahmed, C. F., Tanbeer, S. K. & Jeong, B. S., 2010. A fast indexed based contiguous sequential pattern mining technique in biological data sequences. Jeju, 2nd International Conference on Emerging Databases (EBD'10).
- [10] Karim Md., R., Rashid, M. M., Jeong, B. S. & Choi, H. J., 2012. An efficient approach to Mining Maximal Contiguous Frequent Pattern from Large DNA Sequence Databases. Genomics & Informatics, 10(1), pp. 51-57.
- [11] Tanvee M. M., Kabeer S. J., Chowdhury T. M., Sarja A. A. and Shuvo Md. T. H. 2013. Mining Maximal Adjacent Frequent Patterns from DNA Sequences using Location Information. International Journal of Computer Applications. Vol. 76 – No. 15.
- [12] Zerín, S. F. & Jeong, B. S., 2011. A Fast Contiguous Sequential Pattern Mining Technique in DNA Sequence Using Position Information. IETE Technical Review, 28(6).
- [13] Bona, M. (2011). *A Walk Through Combinatorics - An Introduction to Enumeration and Graph Theory*. World Scientific