International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:2, No:6, 2008

Constraint Based Frequent Pattern Mining Technique for Solving GCS Problem

First G.M. Karthik and Second Ramachandra.V.Pujeri, Dr.

***Abstract*—** Generalized Center String (GCS) problem are generalized from Common Approximate Substring problem and Common substring problems. GCS are known to be NP-hard allowing the problems lies in the explosion of potential candidates. Finding longest center string without concerning the sequence that may not contain any motifs is not known in advance in any particular biological gene process. GCS solved by frequent pattern-mining techniques and known to be fixed parameter tractable based on the fixed input sequence length and symbol set size. Efficient method known as *Bpriori* algorithms can solve GCS with reasonable time/space complexities. *Bpriori* 2 and *Bpriori* 3-2 algorithm are been proposed of any length and any positions of all their instances in input sequences. In this paper, we reduced the time/space complexity of *Bpriori* algorithm by Constrained Based Frequent Pattern mining (*CBFP*) *technique* which integrates the idea of Constraint Based Mining and FP-tree mining. *CBFP* mining technique solves the GCS problem works for all center string of any length, but also for the positions of all their mutated copies of input sequence. *CBFP* mining technique construct TRIE like with FP tree to represent the mutated copies of center string of any length, along with constraints to restraint growth of the consensus tree. The complexity analysis for Constrained Based FP mining technique and *Bpriori* algorithm is done based on the worst case and average case approach. Algorithm's correctness compared with the *Bpriori* algorithm using artificial data is shown.

***Keywords*—** Constraint Based Mining, FP tree, Data mining, GCS problem, *CBFP* mining technique.

I. Introduction

GENERALIZED *Center String problem* [43] is generalized from *CAS* and its variants, where center strings of any length *l* are searched in *N* input sequence of each length of *L* and mutated copy of each center string $(1 < l \le L)$ is contained in at least *q*. *GCS* is more generalized from *CAS*. *Common approximate substring(CAS)* problem is defined as finding a string $t \in \sum^l$ such that for every string s_i in *S*, there exists a substring t_i in s_i $d(t, t_i) \le d$ [38], in a set of sequences $S = \{S_1, S_2, \ldots S_N\}$ over a symbols set $\sum$ such that $|S_i| \le L$,$1 < i \le N$ and positive integers *l* and *d* such that $(1<l\le L)$ and $(0 \le d \le l)$, where $d(t, t_i)$ means the Hamming distance between the string *t* and t_i and substring of length *l* [36]. A string satisfying the objective of *CAS* is often called a *center string*. *CSSP and CSP* are variants of *CAS* are discussed briefly about their role in molecular biology [1], [12], [19], [24], [25], [33]. *CAS* is proven to be *NP-complete* [23], [24] [33]. As for *CAS* and its variants, specifically *CSSP* (*Closet Substring Problem*) and *CSP* (*Closest String Problem*) [1], [12], [19], [24], [25], [33], have vital role in molecular biology in [6], [7], [26], [35], [47], and known as *FPT* (fixed-parameter tractable) with respect to size of symbol set ∑ and length of center string ***l*** [12], [36]. A *center string* is a model for common substring and is not necessarily included in any of the input sequences [43]. Exact solution is given by [5], [21], [27], [28], [31], [45] and [46] to find the center string of length *l* over ∑ concerning all theoretical issues of computational molecular biology. Finding longest center string without concerning the sequence that may not contain any motifs is not known in advance in any particular biological process. This motivated *Ruqian Lu* to generalize *CAS* to *GCS* (Generalized Center String) problem. *Ruqian Lu* proved *GCS* is *FPT* with respect to sequence length *L* and symbol set size |∑| and solved *GCS* with three versions of *Bpriori* (*Biological Variation of the Basic idea of Apriori algorithm*) algorithm.

In this paper we develop and integrate two techniques in order to solve the time and space complexity of the *Bpriori* algorithm [43]. First, a novel, compact *FP-tree* like *TRIE* data structure called *consensus tree* is constructed [4], [14], [17], [18], [29], [37], [42], which is extended prefix-tree structure storing crucial, quantitative information about the frequent pattern. The consensus tree is compact and informative, every node points in the tree consists of strings that occur in more than *q* input sequence with up to certain mutation, each frequent itemset is represented as path from root to some leaf in the tree. The *FP-growth* method is efficient and scalable for mining both long and short frequent patterns [15], [37], and is about an order of magnitude faster than the *Bpriori* algorithm [14], [37], [43]. To ensure that the *FP-growth* method transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent item as a suffix, offering good selectivity.

Second, using the strategy of *Constraint-Based mining* [8], [20], [16], [30], [40], [48], we restraint growth of *FP-tree like TRIE* using the user-specified constraints [40]. *Constraint-based mining* allows us to focus on restraining the growth of consensus tree by providing additional mining constraints. Two category of constraints such as *level constraint* [16] and *rule constraint* [48] (i.e., antimonotonic and succinct) in which *rule constraints* are focused to prune the search space of the FP tree [41], [17], [4]. Integrating two techniques and developed a new

algorithm known as *CBFP (Constraint Based FP tree mining)* to solve *GCS* (*Generalized Center String*) problem with reasonable time and space complexities. *CBFP* algorithm is fixed anchored in fixed symbol set size and fixed length of input sequence. *CBFP algorithm* (similar to *Bpriori* algorithm [43]), proposed based on the downward closure property which narrow down the search space be level-wise search strategy [22].*CBFP algorithm* is proposed based on two points, the first we search for all center strings of any length among input sequences (is a model for common substrings and is not necessarily included in any input sequences [43]). The second one is that we search for all instances of all center string in the input sequences (including de-generative).This paper proves that the goals are same and results can interpreted in both *CBFP algorithm* and *Bpriori algorithm* for *GCS* problem. Implicit user-defined constraint play vital role in pruning the search space of the *FP-tree* and reduce working time.

The remainder of the paper is organized as follows. Section 2 surveys related work. Section 3 discusses the two *Bpriori* algorithms and its operation. *CBFP* tree based frequent pattern mining algorithm is introduced in Section 4. Finally, Section 5 presents some experimental results. Conclusions are given in Section 6.

II. Previous work

Ruqian Lu has generalized CAS to more general problem, GCS, which is FPT with respect to the length of sequences, *L*, and the size of alphabet $|\sum|$. *Bpriori* 1, *Bpriori*2 and *Bpriori*3 are presented for solving GCS [43]. CAS and its variants are proven to be NP-complete [23], [24], [33], and FPT with respect to the size of symbol set $|\sum|$ and *l* length of the string [36]. CAS solved by polynomial-time approximation schemes (PTAS) and for its variants, CSP solved by polynomial-time $(4/3+\varepsilon)$-approximation algorithm and for CSSP, polynomial-time $(1+\varepsilon)$ approximation algorithm given in [19], [25]. To find all exact solution to enumerate all possible $|\sum|^l$ string of length *l* over $\sum$ in computational molecular biology solved by [5], [21], [27], [28], [32], [43], [45], [46]. The variants of CAS, CSP and CSSP are solved by Gramm et al [12]. Sagot [28] and Guan et al [13] gave efficient solution for repeated motif and common motif problem, similar to CAS. Planted motif problem also similar to CAS solved by many algorithms to find motifs up to *d* mismatches [2], [3], [5], [9], [10], [11], [32], [34], [43], and [44].

III. Bpriori algorithm

Bpriori algorithm is introduced to solve the GCS problem [43]. *Bpriori* name reminds of the biological variation of the basic idea of the Apriori algorithm. Three version of *Bpriori* algorithm for solving GCS based on consensus tree with breadth-first and heuristic pruning techniques. The algorithm uses the basic idea and techniques of Apriori algorithm and FP tree method. Apriori algorithm employs *level-wise* and explore $(k+1)$-*itemsets* based on the downward closure property (a set with property *P* is said to satisfy downward closure property if all nonempty subsets of the set must also have the property *P*) [38], [39]. In FP tree (frequent pattern tree) technique, each frequent itemset is represented as a path of a tree, from root to some leaf node, in a way such that more frequently occurring items will have a better chance of sharing nodes than less frequently occurring items [15]. The GCS problem require mining *center string* by finding all mutated copies of *center string* (also called consensus string) limited by $d \geq 0$ among given *N* input sequences. GCS also had downward closure property and use *level-wise* search strategy for finding longer and longer consensus strings.

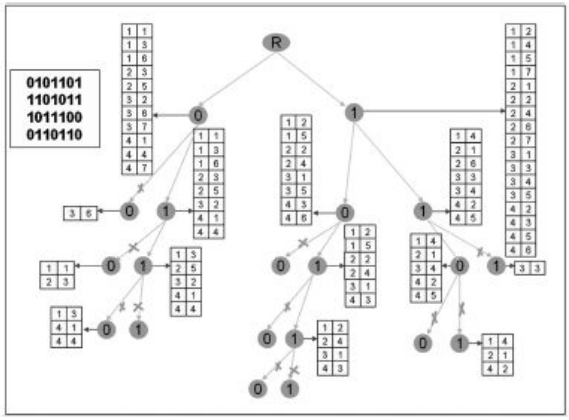


Fig. 1 An example of consensus tree with $d=0, \sum=\{0,1\}$ and $S=\{0101101,1101011,1011100,0110110\}$. $N=4, L=7$ and $|\sum|=2$, $q=4, d=1$

Bpriori uses TRIE like structure (called consensus tree) for shared representation of all consensus strings and a *level-wise* search strategy for solving GCS problem. *Bpriori* algorithms find all *center strings* $t \in \sum^l$ with any length *l*, $0 \leq d < l \leq L$ such that for each *t*, there are at least *q* sequences of *S* containing an *x*-mutated copy $(x \leq d)$ of t along with their position. Three version of *Bpriori* algorithm are *Bpriori*1 for finding *center string* without mismatched $(d=0)$, *Bpriori*2 for finding *center string* with $d \geq 0$ and *Bpriori*3 – space saving algorithm considering the mismatches. Concerning *Bpriori*2 and *Bpriori*3 both construct the TRIE like data structure for consensus tree.

In consensus tree constructed by *Bpriori*1 algorithm shown in Fig. 1, there are $|\sum|$ branches grown out from each non-leaf node *n.* Each substring *t* mapped to each input sequence *s* to a path starting from the root of the tree. Each node n contains pointers to all substrings mapped to *n*, where a pointer (j,k) points to a substring starting at the *k*th position of the *j*th sequence and node containing pointers pointing to less than *q* input sequences will not have no successors. Seven characteristics are stated for the *Bpriori*1 algorithm [43].

Algorithm *Bpriori*1
1. **for** *j* = 1 to *N* **do**
2. **for** *k* = 1 to *L* **do**

3. **if** the *k*th element of the *j*th sequence is $b_1 \in \sum$ **do**
4. put (j,k) in S_{b1} and j in T_{b1}
5. $i \leftarrow 1$;
6. **while** $i < L$ **do begin**
7. **for** each $S_{b_1 b_2 \ldots b_i} \neq \phi$ **do**
8. **for** each (j,k) of $S_{b_1 b_2 \ldots b_i}$ while $k < L-i+1$ **do**
9. **if** the $k+i$ th element of the *j*th sequence is $b_{i+1} \in \sum$ and $|T_{b_2 \ldots b_i b_{i+1}}| \geq q$ **do**
10. put (j,k) in $S_{b_1 b_2 \ldots b_i b_{i+1}}$ and j in $T_{b_1 b_2 \ldots b_i b_{i+1}}$;
11. Remove all $S_{b_1 b_2 \ldots b_i b_{i+1}}$ and $T_{b_1 b_2 \ldots b_i b_{i+1}}$ for $|T_{b_1 b_2 \ldots b_i b_{i+1}}| < q$;
12. **if** all $S_{b_1 b_2 \ldots b_i b_{i+1}}$ are removed then stop the program;
13. Output all pairs $(b_1 b_2 \ldots b_{i+1}, S_{b_1 b_2 \ldots b_i b_{i+1}})$
14. Remove all $S_{b_1 b_2 \ldots b_i}$ and $T_{b_1 b_2 \ldots b_i}$;
15. $i \leftarrow i+1$ **;**
16. **end**

*Bpriori*2 algorithm finds the center string with $d > 0$,in consensus tree shown in Fig. 2 the consensus string are put in the node with *d'* of mismatches, $0 < d' \leq d$ refers to possible mutated copies of the *center string*. *Bpriori*2 generate all *center string* of any length and proved that GCS problem is fixed parameter tractable with respect to the parameter group (l,d) with finite and fixed symbol set $\sum$. *Bpriori*2 has a reasonable time complexity, but an unsatisfying space complexity.

Algorithm Bpriori2

1. **for** *j* = 1 to *N* **do**
2. **for** *k* = 1 to *L* **do**
3. **if** the *k*th element of the *j*th sequence is $b_1 \in \sum$ put $(j,k,0)$ in S_{b_1} , $(j,k,1)$ in all $S_{b_1'}$ for $b_1' \neq b_1$ and j in $T_{b_1''}$ for each $b_1'' \in \sum$
4. $i \leftarrow 1$;
5. **while** $i < L$ **do**
6. **begin**
7. **for** each $S_{b_1 b_2 \ldots b_i} \neq \phi$ **do**
8. **for** each (j,k,e) of $S_{b_1 b_2 \ldots b_i}$ while $k < L-i+1$ **do**
9. **if** the $k+i$ th element of the *j*th sequence is $b_{i+1} \in \sum$ and $|T_{b_2 \ldots b_i b_{i+1}}| \geq q$ **do**
10. **begin**
11. put (j,k,e) in $S_{b_1 b_2 \ldots b_i b_{i+1}}$ and j in $T_{b_1 b_2 \ldots b_i b_{i+1}}$;
12. **if** $e < d$ then for all $b_{i+1}' \neq b_{i+1}$ put $(j,k,e+1)$ in $S_{b_1 b_2 \ldots b_i b_{i+1}'}$ and j in $T_{b_1 b_2 \ldots b_i b_{i+1}'}$;
13. **end;**
14. Remove all $S_{b_1 b_2 \ldots b_i b_{i+1}}$ and $T_{b_1 b_2 \ldots b_i b_{i+1}}$ for $|T_{b_2 b_2 \ldots b_i b_{i+1}}| < q$;
15. **if** all $S_{b_1 b_2 \ldots b_i b_{i+1}}$ are removed then stop the program **else** output all pairs $(b_1 b_2 \ldots b_{i+1}, S_{b_1 b_2 \ldots b_i b_{i+1}})$
16. Remove all $S_{b_1 b_2 \ldots b_i}$ and $T_{b_1 b_2 \ldots b_i}$;
17. $i \leftarrow i+1$ **;**
18. **end;**

*Bpriori*3-1 produces all center string with length ≥ 2, performing many comparison operations for calculating Hamming distance between two strings. *Bpriori*3-1 is good in space complexity, but not as good as *Bpriori*2 in time complexity, provided that this is the lowest space complexity for solving GCS problem. In these mutated copies of all substrings as candidates of *center strings* is not generated. Rather, each center *string* is in the *d-neighborhood* of at least one consensus string is generated and tested along the path in the consensus tree by calculating Hamming distance between two strings. Tree nodes pointing to less than *q* input sequences will not be pruned, because they may be members of *d-neighborhoods* of some potential *center strings*, pruned if the downward closure property is applicable. The consensus tree stops to grow only if at some level of the tree no *center string* is found.

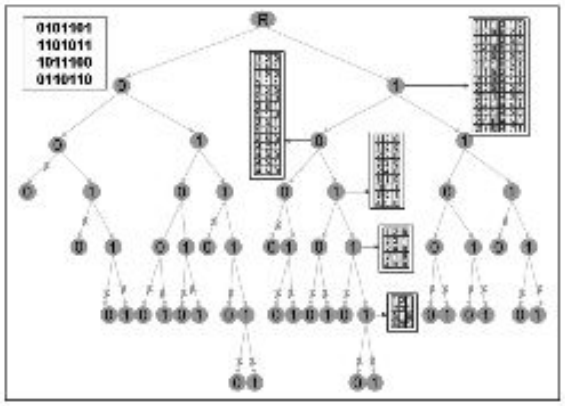


Fig. 2 An example of c onsensus tree with $d = 1 . \sum = \{0,1\}$ and $S = \{0101101, 1101011, 1011100, 0110110\}$. $N = 4, L = 7$ and $|\sum| = 2, q = 4, d = 1$

Algorithm *Bpriori*3-1

1. **for** *j* = 1 to *N* **do**
2. **for** *k* = 1 to *L* **do**
3. **if** the *k*th element of the *j*th sequence is $b_1 \in \sum$,put (j,k) in S_{b_1} ,*j* in T_{b_1} & 1 in R_{b_1} ;
4. $i \leftarrow 1$;
5. **while** $i < L$ **do**
6. **begin(0)**
7. **for** each $R_{b_1 b_2 \ldots b_i} = 1$ **do**
8. **for** each (j,k) of $S_{b_1 b_2 \ldots b_i}$ while $k < L-i+1$ **do**
9. **if** the $k+i$ th element of the *j*th sequence is $b_{i+1} \in \sum$ and $S_{b_2 \ldots b_{i+1}} \neq \phi$, put (j,k) in $S_{b_1 b_2 \ldots b_i b_{i+1}}$ and j in $T_{b_1 b_2 \ldots b_i b_{i+1}}$;
10. **for** each $S_{b_1 b_2 \ldots b_i} \neq \phi$ **do**
11. **for** each $b_1' b_2' \ldots b_i'$ with distance $(b_1 b_2 .. b_i, b_1' b_2' . b_i') \leq d$ **do**
12. **begin (1)**

13. $include \leftarrow \phi$
14. **for** each $S_{b_1''b_2''\ldots b_i''} \neq \phi$ with distance $(b_1''b_2''..b_i'', b_1'b_2'..b_i') \leq d$ **do**
15. $include \leftarrow include \bigcup T_{b_1''b_2''\ldots b_i''}$;
16. **if** $|include| \geq q$ then for all $T_{b_1''b_2''\ldots b_l''} \in include$ **do**
17. **begin (2)**
18. output the pair $(b_1'b_2'..b_i', S_{b_1''b_2''..b_i''})$;
19. $R_{b_1''b_2''..b_i''} \leftarrow 1$;
20. **end (2)**
21. **end (1)**
22. **if** no output is produced then stop the program.
23. remove all $S_{b_1b_2\ldots b_i}$ and $T_{b_1b_2\ldots b_i}$;
24. $i \leftarrow i+1$;
25. **end (0)**

IV. Constraint Based Frequent Pattern Mining Technique

In *Bpriori*2 and *Bpriori*3-1 algorithm solve GCS problem with the consensus tree and a *level-wise* strategy. In *Bpriori*2 maintains the whole tree in the memory, may become crucial at each level by the number of developed nodes. Whereas *Bpriori*3 solve the space complexity of Bprioir2 using weaker pruning policy performs many comparisons for calculating hamming distance between two strings. In this new algorithm, C*onstraints Based Frequent Pattern mining* techniques utilizing the TRIE like structure of FP-tree along with two constraints [16], [20], [30], [40], [48] to solve the GCS problem. *CBFP* technique uses formulation and definition of GCS (*Generalized Center* String) problem mentioned in [43] which generalize the CAS (*Common Approximate* Substring) problem to GCS problem.

Definition 1.*Given* $d \geq 0$ *as the number of maximally allowed mutations (errors or mismatches), any string b with* $d_H(a,b) = x \leq d$ *is called a x-mutated copy (or simply mutated copy) of a and vice versa. A zero mutated copy is also called an exact copy. All x-mutated copies of a, where* $x \leq d$ *, form the d-neighborhood of a. a is called the center of this neighborhood.*

Definition 2.*Given parameters N, L, q, d and* $\sum$ *of the GCS problem, a is called a center string if each of at least q input sequences contains a substring in a's d-neighborhood.*

The formal definition of GCS [43] is, a set $S = \{s_1, s_2, \ldots s_N\}$ of sequences over a finite symbol set $\sum$ with $|\sum| = R$, such that $|s_i| = L, 1 \leq q \leq N$, and positive integers d and q such that $0 \leq d < L$ and $1 \leq q \leq N$.The objective is finding all *center string* $t \in \sum^l$ with any length l, $o \leq d < l \leq L$ *such that for each t*, there are at least q sequences of S containing an *x-mutated* copy $(x \leq d)$ of t along with their positions.

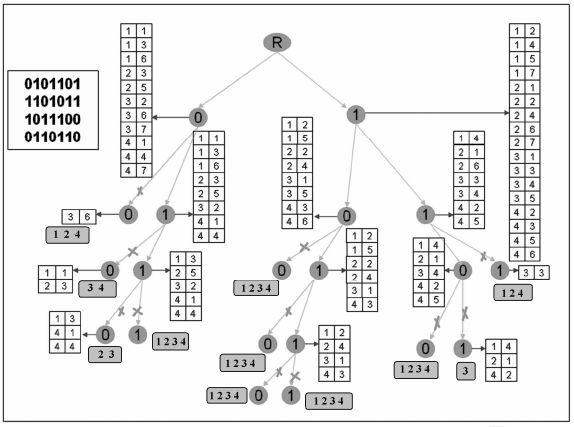


Fig. 3 An example of consensus tree with $d = 0. \sum = \{0,1\}$ and $S = \{0101101, 1101011, 1011100, 0110110\}$. $N = 4, L = 7$ and $|\sum| = 2, q = 4$ using *Bpriori*2 algorithm

CBFP mining technique mine *center string* from given *N* input sequences, finds all of their mutated copies limited by a distance $d \geq 0$ by a *level-wise* search strategy [8],[20]. *CBFP* mining technique constructs a TRIE-like structure for all consensus strings. Consensus tree have $|\sum|$ branches in each level for each non-leaf node *n*. Each node has $|\sum|$ branches only if nodes satisfy prescribed *support* and *confidence* level. Each node in consensus tree contains position for mutated copies of substring $d \geq 0$, a pointer (j,k,e) points to a substring at *j*th position in *k*th input sequence among given *N* input sequences with level of mutation $e \leq d$. A path from root to any node in the consensus tree represents a *center string*. Based on the constraints, pre-pruning the nodes happens at each level like backward closure property. The number of levels of the tree is at most *L* of the sequence. Nodes with *confidence* value $conf(b) = ((N - \sup(b))/(N-q)) < 1$ will be pruned; it is an *antimonotone constraints.* A *constraint P is anti-monotonic if and only if* $\forall$ *sequence x:* $(x \subseteq y) \wedge p(y) \rightarrow p(x)$ [48]. The path from root to a node in the consensus tree represents a mutated copy of a *center string*, and its *confidence* value less than 1 then there is no or less than *q* sequence containing *center string* among given *N* input sequences. The node to be pruned it means further superset belongs to consensus center string is reluctant by definition of GCS. For each node *support* value will be calculated for the number of sequence that does not contribute in production of *center string*; this constraint is *monotonic* in *CBFP mining* technique. A *constraint P is monotonic if and only if* $\forall$ *sequence x:* $(x \subseteq y) \wedge p(x) \rightarrow p(y)$ [48]. The pointers of each node belongs to different genesis that do not contribute to the production of *center string* are counted for all node. *Support* values anticipated to be $\leq q$ for the entire pointers in a node that branches out with $|\sum|$ child, pursue it as monotonic property for all nodes in the consensus tree. Each position of consensus node with level of mutation $e > d$ will be pruned, sustained all position in consensus

node like *succinct* constraint. A *constraint P is succinct, enumerates all and only those sets that are guaranteed to satisfy the constraint* [16]. For each (j,k,e) pointers of consensus node with the mutation level $e > d$ are pruned from the process of production of center string. The mutated copies of all center string with $e > d$ are not be concerned by formulation of GCS [43]. In *CBFP mining* technique, consensus tree's node contains pointer to all its consensus strings, the node pointing to different origins is greater than *q* sequences will not be pruned. The consensus tree is not fully grown, because any node not satisfying *confidence* level (or less than *q* sequence) will not have $|\sum|$ branches, stops to grow at a level of the tree no *center string* is found.

In constrain based mining, constraints are categorized into five constraints [16][48] in which *CBFP mining* techniques uses *antimonotone, monotonic* and *succinct* constraints which meant to restraint the growth of consensus tree with reasonable time and space complexity.

Algorithm *CBFP mining technique*

1. **for** $j = 1$ to N **do**
2. **for** $k = 1$ to L **do**
3. **if** the kth element of the jth sequence is $b_1 \in \sum$ **do**
4. put $(j,k,0)$ in S_{b_1}, $(j,k,1)$ in all $S_{b_1'}$ for $b_1' \neq b_1$ for each $b_1'' \in \sum$ **else** $sup(b_i)$
5. $i \leftarrow 1$;
6. **while** $i < L$ **do**
7. **begin**
8. **for** each $conf(b_1, b_2, ...b_{i+1}) \geq 1$ **do**
9. **begin (3)**
10. **for** each (j, k, e) of $S_{b_1 b_2 .... b_i}$ while $k < L - i + 1$ **do**
11. **begin (2)**
12. **if** the $k+i$ th element of the jth sequence is $b_{i+1} \in \sum$ and $sup(b_{i+1}) < q$ **do**
13. **begin(1)**
14. put (j,k,e) in $S_{b_1 b_2 ....b_i b_{i+1}}$;
15. **if** $e < d$ then for all $b_{i+1}' \neq b_{i+1}$ put $(j,k,e+1)$ in $S_{b_1 b_2 ....b_i b_{i+1}'}$;
16. **end(1);**
17. **end(2);**
18. **if** $conf(b_{i+1}) < 1$ **then** Remove $S_{b_{i+1}}$ **;**
19. **end (3);**
20. **if** all $S_{b_1 b_2 ....b_i b_{i+1}}$ are removed then stop the program **else** output all pairs $(b_1 b_2 .. b_{i+1}, S_{b_1 b_2 ....b_i b_{i+1}})$
21. Remove all $S_{b_1 b_2 .... b_i}$ and $T_{b_1 b_2 ... b_i}$;
22. $i \leftarrow i + 1$ **;**
23. **end;**

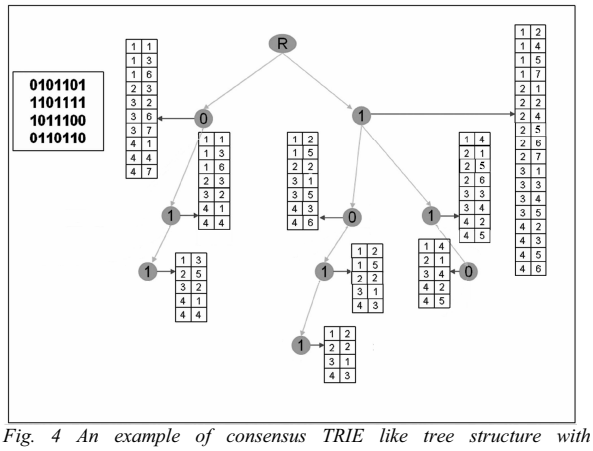
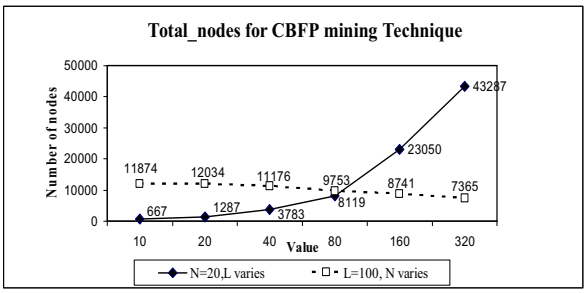


Fig. 4 An example of consensus TRIE like tree structure with $d=0, \sum=\{0,1\}$ *and* $S=\{0101101, 1101111, 1011100, 0110110\}$ $N=4, L=7$ *and* $|\sum|=2, q=4$ *with all pruned nodes created by CBFP mining technique.*

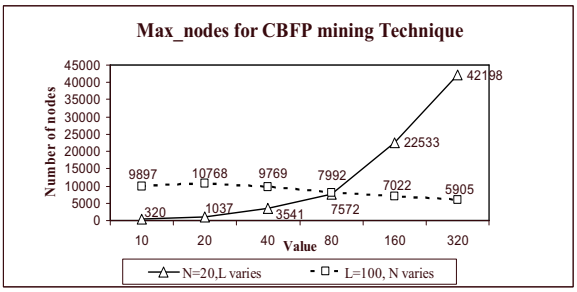
Bpriori algorithm maps each substring of input sequence s to path starting from the root of the tree. Each node n contains pointers to all substring of different origin is less than q sequence will be pruned. But *CBFP* mining technique when node grown out with $|\sum|$ branches will check for $sup(b) \leq q$ and $conf(b) \geq 1$ then node pruned in the middle. *Bpriori* checks whether each node pointers to at least *q* sequences in Fig. 3, rather in *CBFP* mining technique the *confidence* value less than 1 then node will be checked for *center string*. The number of superfluous pointer checking and discarded node formation are proscribed in prior stage than *Bpriori* algorithms. The constraints prescribed in *CBFP* mining technique not at all amend its scenery. In *CBFP* the *support value* can achieve the pruning task but *confident value* make sure for each node creation based on formulation of GCS. Consensus tree is not fully grown in *CBFP* mining technique, restrained by constraint during the mining *center string.*

V. Experiment and Results

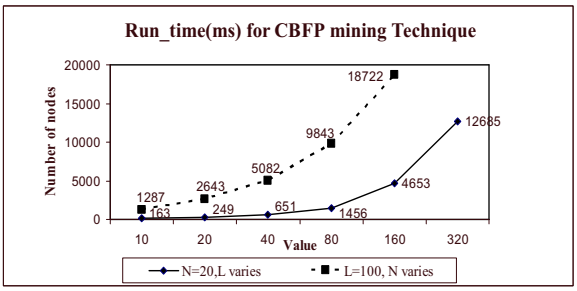
CBFP mining techniques are tested with some imaginary sequences, which are compared with *Bpriori* algorithms. Alike *Bpriori* performance estimation, *CBFP* mining techniques performance evaluated with following parameters Total_nodes, Max_nodes, Run_time(ms) and Real_signal as in [43]. The parameter Total_nodes means the total number of nodes generated by the algorithm, Max_nodes means the maximal number of nodes ever residing in the virtual memory when the algorithm runs,Run_time denotes the running time of the algorithm [43]. Sequences are generated varying the value of the L (length of sequence), N (number of sequence), *q* (maximally allowed sequences) and *d* (mutation level). The worst case analysis and average case analysis are made between the *CBFP* mining technique and *Bpriori* technique by increasing the order of growth of the terms *N* (number of sequences) and *L* (Length of each sequences).



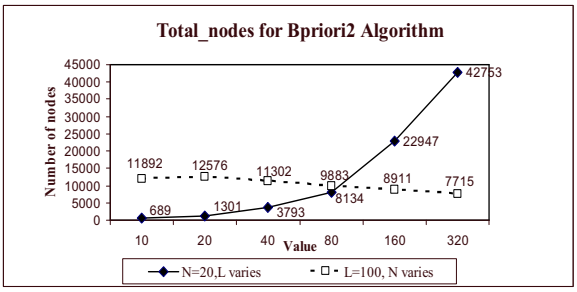
(a)



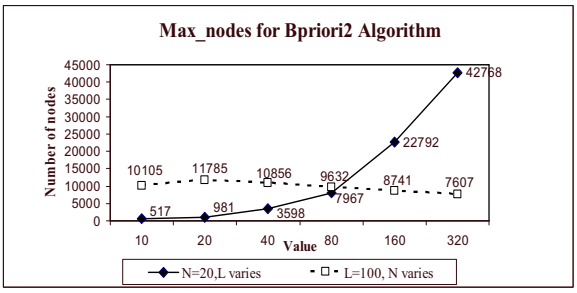
(b)



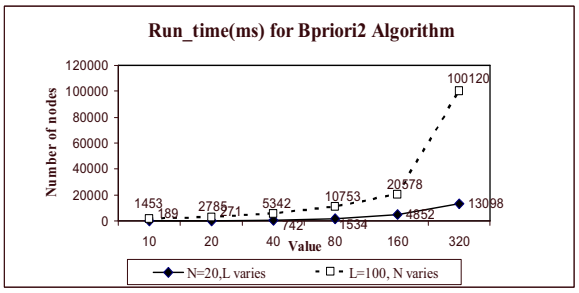
(c)



(d)



(e)



(f)

Fig 5 Graph comparison between the CBFP mining technique(a),(b),(c) & Bpriori2 algorithm(d),(e),(f) for different parameter values N and L test on the condition q= 3N/4 and d=2. In graph solid line represent the parameter N is fixed but the length of sequences L is increased steadily and dashed line represents the parameter L is fixed but the total number of sequences is enlarged gradually.

TABLE I
PERFORMANCE OF BPRIORI 3-1 ALGORITHM

N	L	Total_nodes	Max_nodes	Run_time(ms)
20	10	67421	287	32451
20	20	75129	432	42364
20	40	220874	1023	150342
20	80	539812	2765	973780
20	160	1876239	5196	13216578
20	320	3234564	9624	19458628
10	100	600123	1726	4631749
20	100	790532	3402	1004487
40	100	1217835	5735	929687
80	100	1994452	9643	912164
160	100	2830482	13983	945273
320	100	3667234	17942	950431

Performance of Bpriori 3-1 tabulated based on the parameter N and L with condition q=3N/4 and d=2. In first subgroup N is fixed and L is increased step by step. In second subgroup L is fixed and number of sequences N is increased gradually.

In Fig. 4 *CBFP* mining techniques shows all the nodes which are pre-pruned not like Bpriori algorithms. Both Bpriori2 and Bpriori3-1 algorithm creates the node first with the parents nodes then check for the GCS constraints. CBFP mining techniques uses the constraints restrict the node creation which does not comply. Fig. 5 shows the comparison between the *CBFP* mining technique and *Bpriori2* algorithm on artificial sample data. The sequence *ABDCABAC* is inserted into $3N/4$ among *N* sequences with *d=2*. Each graph consists of two lines, dashed line represents *L* the length of the sequence is fixed to be 100bp and *N* the number of sequence is enlarged from 10 to 320 step by step; solid line shows the number of sequence *N* to be fixed with 20 and the length of sequence *L* is increased gradually. In Fig.5 (a) & (d) illustrate the parameter Total_node, (b) & (e) demonstrate the parameter Max_nodes and (c) & (d) shows Run_time in milliseconds. According to Table I show the performance of the *Bpriori*3-1 algorithm. From the Fig. 5, it is easy to see that,

TABLE II
PERFORMANCE COMPARISON

q	d	Total_nodes	Max_nodes	Run_time(ms)
15	0	68	58	38
		69	64	47
		67	64	65
	1	1056	955	267
		1072	979	328
		19455	1096	4485
	2	11976	9342	2109
		12798	11400	2734
		835008	3243	1193015
2	20	3834	2895	1523
		3901	3527	1797
		500505	2316	166844
	15	12237	10913	2479
		12798	11400	2734
		835008	3243	1193015
	10	25733	20934	2995
		27804	22298	3406
		1321235	3605	1628600

The value of parameter q is fixed and the parameter d is varied in the first subgroup, the parameter d is fixed while the parameter q is varied gradually in the second subgroup on CBFP mining technique, Bpriori2 and Bpriori3-1 algorithm.

the enlargement of the values of *N* and *L,* the running time of *CBFP* is less than the *Bpriori*2 and both increases slowly, but running time for *Bpriori* 3-1 increases fast and space requirement has almost no change with varying increase in *N* and *L*. Increase in *N* and *L* the space requirement for *CBFP* is moderated, *Bpriori* 2 increases fast.

In [43], performance of the algorithms is influenced by the parameter *d*. According to Table 2 the three algorithms performance is compared by varying the parameter *d* and *q* on the synthetic data set with *L=100* and *N=20.* For larger value of *d* the number of node generated is greater but increasing the value of *q* less number of nodes will be produced.

VI. Conclusion

In this paper, GCS problem solved by Constraint based frequent pattern mining techniques. C*onstraints Based Frequent Pattern mining* techniques utilizing the TRIE like structure of FP-tree along with constraints. *CBFP mining* techniques uses *antimonotone, monotonic* and *succinct* constraints which meant to restraint the growth of consensus tree with reasonable time and space complexity. In *CBFP* the *support value* can achieve the pruning task but *confident value* make sure for each node creation based on formulation of GCS. Consensus tree is not fully grown in *CBFP* mining technique, restrained by constraint during the mining *center string. CBFP* mining techniques are tested with some imaginary sequences, which are compared with *Bpriori* algorithms. Time complexity of *CBFP* mining technique is best among some advantageous situations. Pre-pruning techniques and *TRIE* structure plays an important role in real biological data. In future, this algorithm is to be applied for some more real problem in molecular biological and plays vital role in biological computation.

Reference

[1] C.N. Meneses, Z. Lu, A.S. Oliveria, and P.M. Pardolos, "Optimal Solutions for the Closest String Problem via Integer Programming," INFORMS J. Computing, vol. 16, no. 4, pp. 419-429, 2004.
[2] E. Eskin and P. Pevzner, "Finding Composite Regulatory Patterns in DNA Sequences," Bioinformatics, pp. 354-363, 2002.
[3] F.Y.L. Chin and H.C.M. Leung, "Voting Algorithms for Discovering Long Motifs," Proc. Third Asia-Pacific Bioinformatics Conf. (APBC '05), pp. 261-271, 2005.
[4] G. Grahne, L. Lakshmanan, and X. Wang, "Efficient mining of constrained correlated sets," In Proc. 2000 Int. Conf. Data Engineering (ICDE'00), San Diego, CA, pp. 512–521, 2000.
[5] G. Pavesi, G. Mauri, and G. Pesole, "An Algorithm for Finding Signals of Unknown Length in DNA Sequences," Bioinformatics, vol. 17, pp. 207-214, 2001.
[6] G. Pavesi, G. Mauri, and G. Pesole, "In Silico Representation and Discovery of Transcription Factor Binding Sites," Brief in Bioinformatics, vol. 5, no. 3, pp. 217-36, 2004.
[7] G.D. Stormo, "DNA Binding Sites: Representation and Discovery," Bionformatics, vol. 16, no. 1, pp. 16-23, 2004.
[8] H. Mannila, H. Toivonen, "Levelwise search and borders of theories in knowledge discovery," Data Mining and Knowledge Discovery 1, pp 241-258, 1997.
[9] H.C.M. Leung and F.Y.L. Chin, "Algorithms for Challenging Motif Problems," J. Bioinformatics and Computational Biology, vol. 4, no. 1, pp. 43-58, 2006.
[10] J. Buhler and M. Tompa, "Finding Motifs Using Random Projections," Proc. Fifth Ann. Int'l Conf. Computational Molecular Biology (RECOMB), pp. 69-76, 2001.
[11] J. Davila, S. Balla, and S. Rajasekaran, "Space and Time Efficient Algorithms for Planted Motif Search," Proc. Int'l Workshop Bioinformatics Research and Applications (IWBRA '06), May 2006.
[12] J. Gramm, R. Niedermeier, and P. Rossmanith, "Exact Solutions for CLOSEST STRING and Related Problems," Proc. 12th Int'l Symp. Algorithms and Computation, pp. 441-453, 2001.
[13] J. Guan, D.Y. Liu, and D.A. Bell, "Discovering Motifs in DNA Sequences," Fundamenta Informaticae, vol. 59, pp. 119-132, 2004.
[14] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," In Proc. 2000 ACMSIGMOD, Int. Conf. Management of Data (SIGMOD'00), Dallas, TX, pp. 1–12, 2000.
[15] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. ACM SIGMOD, pp. 1-12, 2000.
[16] J. Han, L.V.S. Lakshmanan, T.N.Raymond, "Constraint-Based Multidimensional Data Mining," IEEE Computer, 32(8), 46-50,1999.
[17] J. Pei, J. Han, and L.V.S. Lakshmanan, "Mining frequent itemsets with convertible constraints," In Proc. 2001 Int. Conf. Data Engineering (ICDE'01), Heidelberg, Germany, pp. 433–332, 2001.
[18] J. Pei, J. Han, and R. Mao, "CLOSET: An efficient algorithm for mining frequent closed itemsets," In Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD'00), Dallas, TX, pp. 11–20, 2000.
[19] K. Lanctot, M. Li, B. Ma, and L. Zhang, "Distinguishing String Selection Problems," Information and Computation, vol. 185, no. 1, pp. 44-51, 2003.
[20] L. De Raedt, S. Kramer, "The levelwise version space algorithm and its application to molecular fragment finding," In. IJCA101: 7th Int. Joint Conf. Artificial Intelligence, 2001.
[21] L. Marsan and M.F. Sagot, "Algorithms for Extracting Structured Motifs Using a Suffix Tree with Application to Promoter and Regulatory Site Consensus Identification," J. Computational Biology, vol. 7, pp. 345-360, 2000.
[22] M. Ester and X. Zhang, "A Top-Down Method for Mining Most Specific Frequent Patterns in Biological Sequence Data," Proc. SIAM Int'l Conf. on Data Mining (SDM '04), Apr. 2004.
[23] M. Frances and A. Litman, "On Covering Problems of Codes," Theoretical Computer System, vol. 30, pp. 113-119, 1997.
[24] M. Li, B. Ma, and L. Wang, "Finding Similar Regions in Many Strings," Proc. 31st Ann. ACM Symp. Theory of Computing (STOC '99), pp. 473-482, 1999.
[25] M. Li, B. Ma, and L. Wang, "On the Closest String and Substring Problems," J. ACM, vol. 49, no. 2, pp. 151-171, 2002.

[26] M. Tompa et al., “Assessing Computational Tools for the Discovery of Transcription Factor Binding Sites,” Nature Biotechnology, vol. 23, no. 1, pp. 137-144, 2005.
[27] M. Tompa, “An Exact Method for Finding Short Motifs in Sequences, with Application to the Ribosome Binding Site Problem,” Proc. Seventh Int’l Conf. Intelligent Systems for Molecular Biology, pp. 262-271, 1999.
[28] M.F. Sagot, “Spelling Approximate Repeated or Common Motifs Using a Suffix Tree,” Proc. Third Latin Am. Symp. Theoretical Informatics, pp. 111-127, 1998.
[29] M.J. Zaki, and C.J. Hsiao, "CHARM: An efficient algorithm for closed itemset mining," In Proc. 2002 SIAMInt. Conf. Data Mining, Arlington, VA, pp. 457–473, 2002.
[30] M.N. Garofalakis, R. Rastogi, and K. Shim, "SPIRIT: Sequential pattern mining with regular expression constraints," In Proc. 25th Int. Conf. Very Large Data Bases (VLDB '99), San Francisco, Morgan Kaufmann, pp 223-234, 1999.
[31] M.S. Waterman, R. Arratia, and D.J. Galas, “Pattern Recognition in Several Sequences: Consensus and Alignment,” Bull. Math. Biology, vol. 46, pp. 515-527, 1984
[32] P. Pevzner and S. Sze, “Combinatorial Approaches to Finding Subtle Signals in DNA Sequences,” Proc. Int’l Conf. Intelligent Systems for Molecular Biology, pp. 269-278, 2000.
[33] P.A. Evans and A.D. Smith, “Complexity of Approximating Closest Substring Problems,” Fundamentals of Computation Theory, pp. 210-221, 2003.
[34] P.A. Evans and A.D. Smith, “Toward Optimal Motif Enumeration,” Proc. Algorithms and Data Structures, Eighth Int’l Workshop (WADS ’03), pp. 47-58, 2003.
[35] P.A. Evans and H.T. Wareham, “Practical Algorithms for Universal DNA Primer Design: An Exercise in Algorithm Engineering,” Currents in Computational Molecular Biology 2001, N. El-Mabrouk, T. Lengauer, and D. Sankoff, eds., pp. 25-26, Les Publications CRM, 2001.
[36] P.A. Evans, A.D. Smith, and H.T. Wareham, “On the Complexity of Finding Common Approximate Substrings,” Theoretical Computer Science, vol. 306, no. 1-3, pp. 407-430, 2003.
[37] R. Agarwal, C. Aggarwal, and V.V.V. Prasad, "A tree projection algorithm for generation of frequentitemsets," Journal of Parallel and Distributed Computing, 61:350–371, 2001.
[38] R. Agrawal and R. Srikant, “Fast Algorithms for Mining Association Rules,” Proc. 20th Int’l Conf. Very Large Data Bases (VLDB ’94), pp. 487-499, 1994.
[39] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and I. Verkamo, “Fast Discovery of the Association Rules,” Advances in Knowledge Discovery and Data Mining, pp. 307-328, AAAI Press, 1996.
[40] R. Ng et al., "Exploratory Mining and Pruning Optimizations of Constrained Associations Rules," Proc. 1998 ACM SIGMOD Int'l Conf. Management of Data, ACM Press, New York, 1998, pp. 13-24.
[41] R. Srikant, and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," In Proc. 5th Int. Conf. Extending Database Technology (EDBT’96), Avignon, France, pp. 3–17, 1996.
[42] R. Srikant, Q. Vu, and R. Agrawal, "Mining association rules with item constraints," In Proc. 1997 Int. Conf. Knowledge Discovery and Data Mining (KDD’97), Newport Beach, CA, pp. 67–73, 1997.
[43] Ruqian Lu, Caiyan Jia, Shaofang Zhang, Lusheng Chen, Hongyu Zhang, "An Exact Data Mining Method for Finding Center Strings and All Their Instances," IEEE Trans. Knowledge and Data Engineering," 19(4), 509-522, 2007.
[44] S. Rajasekaran, S. Balla, and C.-H. Huang, “Exact Algorithms for the Planted Motif Problem,” J. Computational Biology, vol. 12, no. 8, pp. 1117-1128, 2005.
[45] S. Sinha and M. Tompa, “Discovery of Novel Transcription Factor Binding Sites by Statistical Overrepresentation,” Nucleic Acids Research, vol. 30, no. 24, pp. 5549-5560, 2002.
[46] S. Sinha and M. Tompa, “YMF: A Program or Discovery of Novel Transcription Factor Binding Sites by Statistical Overrepresentation,” Nucleic Acids Research, vol. 31, pp. 3586-3588, 2003.
[47] S.T. Jensen, X.S. Liu, Q. Zhou, and J.S. Liu, “Computational Discovery of Gene Regulatory Binding Motifs: A Bayesian Perspective,” Statistical Science, vol. 19, pp. 188-204, 2004.
[48] Sau Dan Lee, Luc De Raedt, "Constraint Based Mining of First Order Sequences in SeqLog," Database Support for Data Mining Applications 154-173, 2004.

G.M. Karthik, Born in Madurai, Tamil Nadu state in India, in 1981, received the B.E. in Computer Science and Engineering from SACS MAVMM Engineering College, Madurai, M.E. in Computer Science and Engineering from PSNA College of Engineering and Technology, Dindugal, in 2003 and 2005 respectively. He is having 3 years of teaching experience in more than two engineering colleges in India. This paper was written while he was working on the project on Data Mining using intelligent techniques as a Research scholar at Anna University, Chennai, India. His primary research interests are related to Data Mining and Web Mining. Currently, he is working as Senior Lecturer of Computer Science Engineering Department of SACS MAVMM Engineering College, Madurai, India.

Ramachandra.V.Pujeri, Born in Bijapur, Karnataka state in India, in 1973, received the B E in Electronics and Communication Engineering from Karnataka University, Dharwad, ME in Computer Science and Engg from PSG College of Technology, Coimbatore, Ph.D in Information and Communication Engineering from Anna University, Chennai, MBA in Human Resource Management, from Pondicherry University, Pondicherry, in 1996, 2002, 2007 and 2008 respectively. He is active life member of ISTE, SSI, MIE, ACS and IEE. His has written three textbooks. He is having around 13 years of teaching experience in the various top ten engineering colleges in India. He is an active expert committee member of AICTE, NBA, DoEACC, NACC and various Universities in India. Currently, under him ten research scholars pursuing their Ph.D. His research interests lie in the areas of Computer Networking, Operating System, Software Engineering, Software Reliability, Modeling and Simulation, Quality of Services and Data Mining. Currently, he is working as Vice-Principal of KGiSL Institute of Technology, Coimbatore.