Journal of Computational Intelligence in Bioinformatics.
ISSN: 0973-385X Volume 3, Number 2 (2010) pp. 239–247

http://www.ripublication.com/jcib.htm

A DNA Algorithmic approach to solve GCS Problem

A. Murugan
Department of Computer Science, Dr. Ambedkar Government College, Chennai - 600 039, INDIA.
E-mail: amurugan1972@gmail.com

B. Lavanya
Department of Computer Science, University of Madras, Chennai - 600 005, INDIA.

Abstract

Generalized Center String(GCS) problem is generalized from Common Approximate Substring problem (CAS) and Common SubString Problems (CSSP). GCS is known to be NP-hard allowing the problem to lie 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 and their number of occurences in the given strings play a vital role in many bio informatical analysis. In this paper, we propose and study GCS problem, where not only all models of any length, but also the positions of all their degenerative instances in input sequences are parallely searched. First, the DNA strands are parallely generated for a given level number. Then based on the DNA strands generated, we propose a DNA priori algorithm, that solves GCS with efficient time complexity. Implementation results shown the correctness of the algorithm and the validity of the complexity analysis.

AMS Subject Classification:
Keywords: Data Mining, DNA computation, GCS, Java threads.

1. Introduction

Generalized Center String (GCS) problem [13] is generalized from CAS, Common Approximate Substring, and its variants, where center strings of any length l are searched

in N input sequences of each length of L and mutated copy of each center string $(1 < i <= L)$ is contained in at least q occurrences. A string satisfying the objective of CAS is often called a *center string*. CSSP and CSP are variants of CAS, discussed in [4, 9, 10, 15, 16, 17] about their role in molecular biology. CAS is proven to be *NP-complete* [14, 15, 17]. As for CAS and its variants, specifically CSSP (Closet Substring Problem) and CSP (Closest String Problem) [4, 9, 10, 15, 16, 17], have vital role in molecular biology [5, 8, 19, 21, 26] and known as FPT (Fixed-Parameter Tractable) with respect to size of symbol set Σ and length of center string l [9, 18]. A *center string* is a model for common substring and is not necessarily included in any of the input sequences [13].

In this paper, we study GCS problem, where not only all models of any length, but also the positions of all their degenerative instances in input sequences, are parallely searched. We have proved the GCS is fixed parameter tractable with respect to fixed symbol set size and fixed length of input sequences.

2. Literature Review

Solutions are given in [2, 3, 12, 20, 25, 27, 28] to find the center string of length l over Σ concerning all theoretical issues of computational molecular biology. Simulation of all the DNA operations are done in [11], the proposed work uses the DNA operations *cut* and *pcr* found in [11]. 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 problem. Ruqian Lu proved GCS is FPT with respect to sequence length L and symbol set size $|\Sigma|$ and solved GCS with three versions of Bpriori (Biological variation of the basic idea of Apriori algorithm) algorithm [13]. CBFP (Constraint Based Frequent Pattern tree) algorithm [7] is used to 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) using $antimonotone$, $monotonic$, and $succint$ constraints. The goals are same for both *CBFP* and *Bpriori* algorithms. Implicit user-defined constraint play vital role in pruning the search spaces of the FP(Frequent Pattern) tree. In FP tree each frequent item set is represented as a path of a tree, from root to some leaf node.

The time complexity of Briori-2 algorithm [13] is

$$O_{(t-Bpriori2)} = O(N \times L \times l^{d+1} \times v(d,l))$$

where

$$v(d,l) = \frac{|\sum|^d}{(d-1)!}$$

and the space complexity is

$$O_{(s-Bpriori2)} = O(N \times L \times v(d,l))$$

2.1. Formulation of GCS

Suppose $\sum$ is a finite set of symbols and $|\sum|$ its cardinality. Consider the problems in molecular biology as an example, for DNA, $\sum = (A, T, C, G)$ are the nucleotides and $|\sum|$ is 4; for proteins, the symbols are the amino acids and $|\sum|$ is 20.

Let $a = a_1, a_2, a_3, \ldots, a_n$ and $b = b_1, b_2, b_3, \ldots, b_n$ be two strings from $\sum^{+}$. The Hamming distance $d_H(a, b)$ between a and b is defined as:

$$d_H(a, b) = \sum \varepsilon(a_i, b_i),$$

where $i = 1$ to n and

$$\varepsilon(x, y) = 1, \quad \text{when } x \neq y \text{ or } 0, \quad \text{otherwise.}$$

It can be interpreted as the number of symbol mutations needed to turn one string into another.

Definition 2.1. Given d ≥ 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, and is called the center of this neighborhood [13].

Definition 2.2. Given parameters N, L, q, and d of the GCS problem, a is called a center string if each of at least q input sequences contains a sub string in a's d-neighborhood [13].

The formal definition of GCS is as follows:

Given A set $S = S_1, S_2, S_3, \ldots, S_n$ of sequences over a finite symbol set $\sum$ with $|\sum| = R$, such that $|S_i| = L$, $1 \leq i \leq n$, and positive integers d and q such that $0 \leq d < L$ and $1 \leq q \leq N$.

It is easy to see that if we give up two requirements of GCS, namely, finding positions of all (degenerative) instances of these center strings, we come back to CAS. Thus, NP-hardness of GCS is straightforward. In addition, GCS becomes a common motif problem if the length of center strings is specified, and repeated motif problem if $N = 1$.

For solving the GCS problem in practice, an exact and efficient pattern enumeration approach is described in the next section.

3. DNA algorithm: DNA priori Algorithm

CBFP, Bpriori2 and Bpriori3-1 algorithms solve GCS problem with the consensus tree and a level-wise strategy. In the proposed method, the nucleotides are encoded to combinations of *0's* and *1's*, for ease of computation, where *AT* represents *0* and *CG* represents

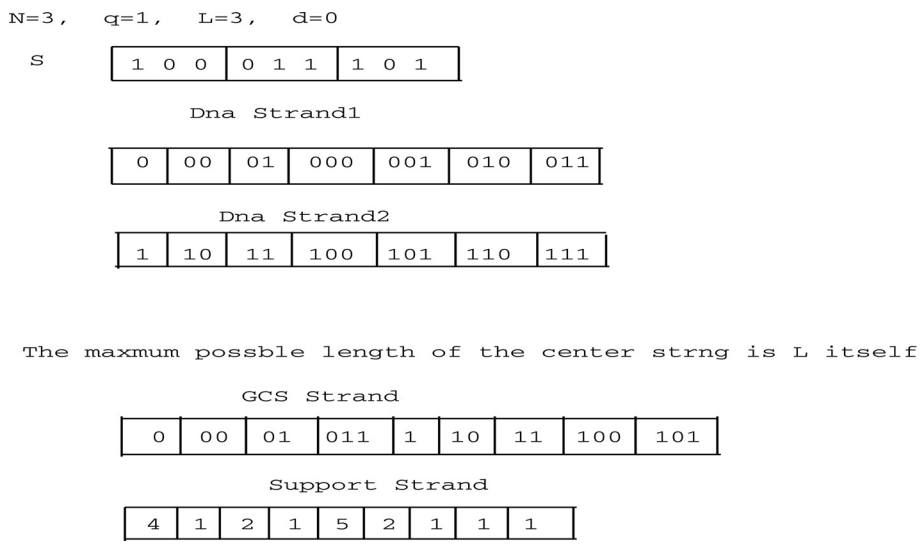

Figure 1: GCS generation

1. The possible center strings are generated and stored in a sequential manner (in DNA strands), the generation is again parallelized by generating two such strands, one for *0* combination of possible center strings and the other strand for *1* combination of possible center strings. We propose a DNA priori algorithm where it generates the DNA strands parallelly based on the given level number, for combinations of *0's* and *1's*, one for each strand. The length of the strand generated is, with respect to the level number. The level number is chosen in such a way that, the maximum length of the possible center string, is the length of the input sequence itself for *0* mutations. Since in DNA computing the storage space is not a constraint, each element is assigned to a separate strand. Each of the strand element is checked, parallely and independently, for its GCS possiblity and its support in, all the given input sequences are counted. If the support is greater than or equal to the given constraint, the element in the strand is a GCS, it is then moved to GCS strand that is the output strand and its respective support count is moved to the support strand. If the equality is not found, the number of mutations are also calculated. An example is illustrated in the Figure 1.

Input for the DNA priori Algorithm 1 are S (the array of DNA input strands), N (number of elements in S), q (user defined minimum support), d (number of mutations allowed), $levelnumber$, represents the length of the center string of the given sequence, which can be maximum length of the input sequence itself for *0* mutations. Algorithm generates DNA strand with GCS elements and its respective support as output. Step1 to Step3 performs all required initialization and assignments. Step 4 and 5 generates the DNA strands with all possible generalised center strings (gcs), with respect to the given level number, that is, the possible gcs for *'0'* combinations are generated and stored in DnaStrand1 and the possible gcs for *'1'* combinations are generated and stored in DnaStrand2. The DnaStrand1 contains all possible center strings of *0's* combination and DnaStrand2 contains possible center strings of *1's* combination as shown in Figure 1. In

Step 7, each element in both the strands, is assigned to separate DNA strand, means that first element of DnaStrand1 to strand1, second to strand2 and so on till the last element in DnaStrand1. Similair process is done for elements in DnaStrand2 also. In Step 8, Threads are created, such that the maximum number of the threads created is equal to the total number of strands, means the total number of possible center strings. Each thread is assigned a strand, say thread1 to strand1, thread2 to strand2 and so on. In Step 9-17, on each strand, parallely and independently, for all N, *pcr* operation [11] is used to multiply the strands for processing, such that the original input is stored. Then the cut operation [11] is performed on the input strings with respect to d mutations, as illustrated in Figure 2. Total number of *strand1* occurences are examined, that is *supp* is counted. In Step 13-16, if $supp$ is $\geq q$, then $strand1$ is added to GCS strand and its $supp$ is added to $support$ strand.

Algorithm 1: DNA priori Algorithm for GCS

Input: $S, levelnumber, user\ defined\ constraint\ q^{+}$, N, d
Output: GCS DNA strand, support DNA strand.
begin
  $let\ L \leftarrow max(length(S(element)))$;
  $let\ supp \leftarrow 0$;
  $let\ DnaStrand1 \leftarrow DNAStrand(PossibleGCSstartswith0)$;
  $let\ DnaStrand2 \leftarrow DNAStrand(PossibleGCSstartswith1)$;
  Generate DnaStrand1, DnaStrand2 parallely for the given *level number* ;
  $let\ strand1 \leftarrow firstelementof\ DnaStrand1$;
  Create threads for each strand ;
  foreach $thread$ **do**
    $[parallely\ for\ each\ thread]$;
    $let\ s1 \leftarrow pcr(S)$;
    $supp \leftarrow cut(s1, strand1, d)$;
    if $supp \geq q$ **then**
      add to GCS ;
      add to support ;
    end
  end
end

+: The minimum required support, that is the minimum number of times, the possible center string, appear in input sequences, for it to qualify as GCS.

Lemma 3.1. Let C be the center string, where $1 \leq |C| \leq |S|$, where S is the given string.

Proof. Obvious. ■

Example 3.2. Let us consider two input sequences of 3 bits each, with minimum user

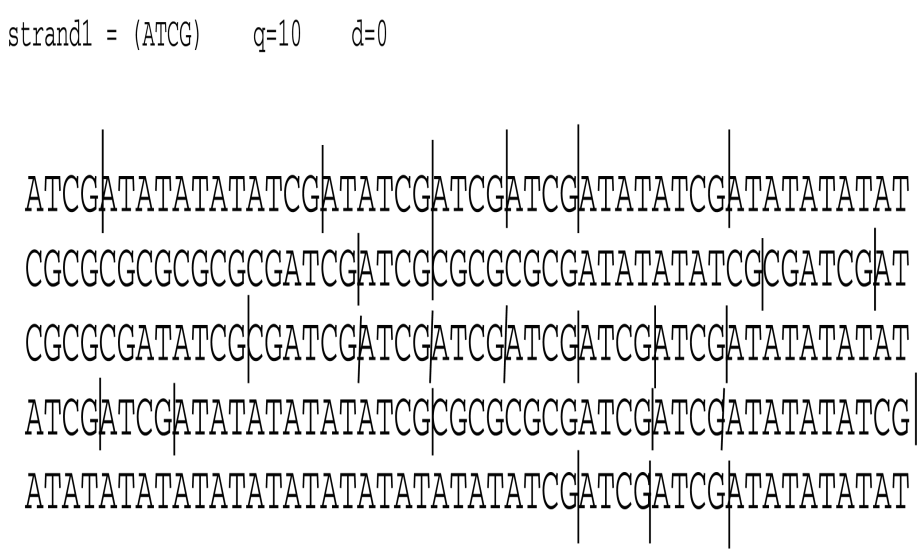

Figure 2: Cutting and Support Counting

defined constraint as one and with zero mutations. Here the level number is chosen to be three as illustrated below:

$$N = 2, q = 1, d = 0, L = 3, \text{ Level number } = 3 \text{ and } S = (100, 110)$$

The possible center strings are 0, 00, 01,000, 001, 010, 011, 1, 10, 11, 100, 101, 110, 111
The GCS strings are 0, 1, 00, 10, 11, 100, 110
The support count is 3, 3, 1, 2, 1, 1, 1
The maximum length of center string is 3 that is the level number (the length of the input sequence) for $d = 0$.

Time Complexity 3.3. The algorithmic steps are parallely executed for all possible center strings, that is, for each element of the DnaStrand1 and DnaStrand2. Thus the time complexity of the algorithm is reduced to the time taken for processing one element from the total number of n elements. This process is explained in the Figure 3. The time complexity of the proposed DNA Algorithm is calculated as follows

$$\begin{aligned}\text{O(t-DNA priori Algorithm)} &= \text{O(Strand Generation)} + \text{O(Checking for support and distance for a strand)} \\ &= O(n/2) + O(N \times L \times n) \\ &= O(n/2) + O(N \times L \times 1)\end{aligned}$$

The space complexity is calculated as

$$\text{O(s-DNA priori Algorithm)} = O(n) + O(N \times L \times n)$$

The proposed algorithm looks for center strings and not for common substrings, no model is needed for the algorithm, it also looks for all possible lengths of center strings, all instances of center strings and their distances, in all the given input sequences.

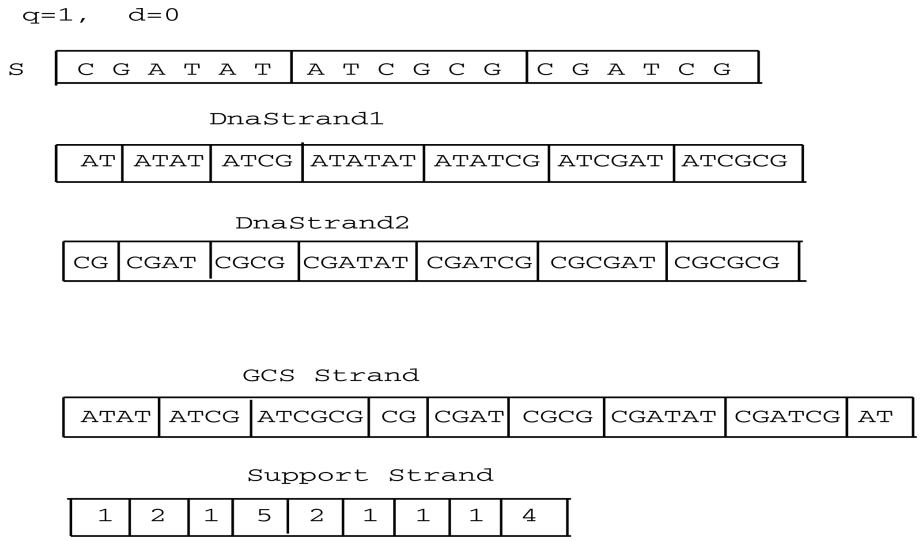

Figure 3: GCS generation with DNA nucleotides

4. Applications

Much data is in sequential format, ranging from purchase histories to program traces, DNA, and protein sequences. In many of these sequential data sources, patterns or behaviors of interests often repeat frequently within the given large sequences. These serve as the motivation for the proposed work. For example, gene regulatory motifs in inter genic DNA sequences play a principle role on the control mechanisms for activating and deactivating the genes [1, 5, 6, 18, 22, 23, 24]. They are short contiguous fragments located in the upstream region of a gene, and have relatively high sequence conservation. In this paper, we have designed and performed the implementation to solve it in a highly parallel way, for finding all subsequences, and can be extended to many other datamining applications also. All implementations are performed on a Pentium IV computer with 1.8 GHz processor and 768MB main memory. The operating system is Windows XP. The resulted data of these experiments are consistent with the complexity analysis given in the previous section.

The proposed work has its importance in solving applications in molecular biology, data mining applications, rule-based systems, genetic algorithms, pattern recognition problems, identify behavioral patterns, episode mining, large set of statistical results analysis, etc, parallely and efficiently.

5. Conclusion and Future Scope

In this paper, we first generalize CAS to a more general problem GCS, which is a fixed - parameter tractable with respect to the length of sequences L, and the size of alphabet $|\sum|$. Then a DNA based algorithm is proposed to solve GCS parallelly, to increase the

efficiency of the motif search. In this, a highly parallel method involving the generation of DNA strands and checking for the existence of center strings for $d = 0$ and $d > 0$, where d is the hamming distance, is proposed. The limitation of this algorithm is that for larger data bases the maximum number of threads generated is dependent on the efficiency of the operating system. In the future, it is possible to apply this approach to solve more real time problems in molecular biology.

References

[1] R. Agarwal, H. Mannila, R. Srikant, H. Toivonen, and I. Verkamo, Fast discovery of the association rules, *Advances in Knowledge Engineering and Data Mining*, 307–328, 1996.

[2] S. Sinha amd M. Tompa, Discovery of novel transcription factor binding sites by statistical overrepresentation, *Nucleic Acid Research*, 30(24):5549–5560, 2002.

[3] S. Sinha amd M. Tompa, Ymf: A program or discovery of novel transcription factor binding sites by statistical overrepresentation, *Nucleic Acid Research*, 31(24):3586–3588, 2003.

[4] C.N. Meneses, Z. Lu, A.S. Oliveria, and P.M. Pardolos, Optimal solution for the closest string problem via integer programming, *INFORMS J. Computing*, 16(4):419–429, 2004.

[5] M. Tompa et al., Assessing computational tools for the discovery of transcription factor binding sites, *Nature Biotechnology*, 23(1):137–144, 2005.

[6] F.Y.L. Chin and H.C.M. Leung, Voting algorithms for dicovering long motifs, *Proc. Third Asia-Pacific Bioinformatics (APBC'05)*, 261–271, 2005.

[7] G.M. Karthik and Ramachandra, V. Pujeri, Constraint based frequent pattern mining technique for solving gcs problem, *Proc. world Academy of Science, engineering and Technology*, 32:672–679, 2008.

[8] S.T. Jensen, X.S. Liu, Q. Zhou, and J.S. Liu, Computational discovery of gene regulatory binding motifs: A bayesian perspective, *Statistical Science*, 19:188–204, 2004.

[9] J. Gramm, R. Niedermeier, and P. Rossmanith, Exact solution for closest string and related problems, 441–453, 2001.

[10] K. Lanctot, M. Li, B. Ma, and L. Zhang, Distinguishing string selection problems, *Information and Computation*, 185(1):44–51, 2003.

[11] B. Lavanya, A. Murugan, and K. Shyamala, Real time environment for dna computing, *Communicated to International Journal of Computer Science*, May 2010.

[12] L. Marsan and M.F. Sagot, Algorithms for extracting structured motifs using a suffix tree with application to a promoter and regulatory site consensus identification, *J. Computational Biology*, 7:345–360, 2000.

[13] Ruqian Lu, Ciayan Jia, Shaofang Zhang, Lusheng Chen, and 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.

[14] M. Frances and A. Litman, On covering problems of codes, *Theoretical Computer System*, 30:113–119, 1997.

[15] M. Li, B. Ma, and L. Wang, Finding similair regions in many strings, *Proc. 31st Ann. ACM Symp. Theory of Computing (STOC'99)*, 473–482, 1999.

[16] M. Li, B. Ma, and L. Wang, On the closest string and substring problems, *J. ACM*, 49(2):151–171, 2002.

[17] P.A. Evans and A.D. Smith, Complexity of approximation of closest substring problem, *Fundamentals of Computational Theory*, 210–221, 2003.

[18] P.A. Evans, A.D. Smith, and H.T. Wareham, On the complexity of finding common approximate substrings, *Theoritical Computer Science*, 306(1-3):407–430, 2003.

[19] P.A. Evans and H.T. Wareham, Practical algorithms for universal dna primer design: An exercise in algorithm engineering, *Currents in Molecular Biology*, 25–26, 2001.

[20] G. Pavesi, G. Mauri, and G. Pesole, An algorithm for finding signals of unknown length in dna sequences, *Bioinformatics*, 17:207–214, 2001.

[21] G. Pavesi, G. Mauri, and G. Pesole. In silico representation and discovery of transcription factor binding sites, *Brief in Bioinformatics*, 5(3):217–236, 2004.

[22] J. Pei, J. Han, and L.V.S. Lakshmanan, Mining frequent itemsets with convertible constraints, *Proc. 2001 Int. Conf. Data Engineering (ICDE '01)*, 433–332, 2001.

[23] J. Pei, J. Han, and R. Mao, Closet, an efficient algorithm formining frquent closed itemsets, *Proc. 2000 ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery (DMKD '00)*, 11–20, 2000.

[24] L. De Raedt and S. Kramer, The levelwise version space algorithm and its application to molecular fragment finding, *In. IJCA101:7th Int. Joint Conf. Artificial Intelligence*, 2001.

[25] M.F. Sagot, Spelling approximate repeated or common motifs using a suffix tree, *Proc. Third Latin Am. Symp. Theoretical Informatics*, 111–127, 1998.

[26] G. D. Stormo, Dna binding sites: Representation and discovery, *Bioinformatics*, 16(1):16–23, 2004.

[27] M. Tompa, An exact method for finding short motifs in sequences with application to ribisome binding site problem, *Proc. Seventh Int'l Conf Intelligent Systems for Molecular Biology*, 262–271, 1999.

[28] M.S. Waterman, R. Arratia, and D.J. Galas, Pattern recognition in several sequences: Consensus and alignment, *Bull. Math. Biology*, 46:515–527, 1984.