BAJOPMAS JOURNAL 16(2): 1 – 14
Printed in Bayero University, Kano - Nigeria.

ALTERNATIVE BIT-PARALLEL STRING PATTERN MATCHING ALGORITHMS FOR BIOLOGICAL SEQUENCES

Muhammad, Y. M.
Department of Computer Science, Faculty of Computing, Bayero University, Kano. Nigeria
mymuhammad.cs@buk.edu.ng

ABSTRACT
The inherent parallelism in a bit operations like AND/OR inside a computer word is known as Bit-parallelism. It plays a greater role in string pattern matching and has good application in the analysis of Biological data. However, benchmark pattern matching algorithms such as Shift-Or or Shift-And lacks the capabilities to be applicable in Biological sequences analysis due to the problem of false matches associated with them. To this end, many Bit-parallel matching algorithms were proposed in recent time to improve upon the performances of the benchmark algorithms and make them more applicable in the Biological sequence analysis. Therefore, this paper presented the working of some of the recently developed Bit-paralleled strings matching algorithms and their application on Biological sequences and also the analysis of their performances to aid Bioinformatics researchers choosing the appropriate algorithm to use.

***Keywords**: Bit-parallelism, Automaton, Pattern matching, Ribonucleic acid,Deoxyribonucleic acid.*

1.0 INTRODUCTION
String analysis is one of the important components of biological research and plays a greater role in the analysis of biological sequences; this is because biological objects can be interpreted as strings. For example DNA is encoded as string forms from four (4) letter alphabets {A, C, G, T} each corresponding to one the DNA nucleotides or bases i.e. (A for adenine), (G for guanine), (C for cytosine) and (T for thymine). RNA can also be encoded as string from four letter alphabets however, RNA has Uracil (U) base instead of thymine in DNA. Protein is encoded as string from twenty (20) letter alphabets corresponding to 20 amino acids making the protein molecule.

One of the string analysis is string pattern matching, a technique that find a copy of one string called a pattern or query from larger string called Text or Reference. String pattern matching is one of the most important tools used by computer science researchers to analyze biological sequences. It is used to find homology between a query sequence whose properties are unknown and sequences of known properties in a database that is critical in inferring the function of newly discovered biological sequences [1].

The use of pattern matching approach has been a greater tool in solving the problems of bio-sequence analysis. It is applicable in finding homology between fully characterize sequence and newly discovered sequence. Researchers have proposed the use of pattern matching methods to solve the problem of proteomic sequence analysis. For example, method that uses the edit distance between bases and complementary base pairs to match RNA secondary structure was proposed in [7]. Another algorithm that finds the exact matching of RNA secondary structure expression was proposed in [8] in $O(nm^2)$ time , where **m** is the size of the secondary expression and **n** is the size of the text. An indexing

approaches discussed in [9] [10] uses a bidirectional affix tree data structure for exact and inexact pattern matching of RNA structure with the worst case time in ***O(nm)*** and ***O(n)*** space where m is the pattern length and n is the database string length, a more efficient method that uses affix array data structure with affix tree functionality and structural suffix tree data structure for RNA based pattern matching problems was studied in [11]. The problem of pattern matching on biological sequence can be formally defined as follows:

Definition1. *Let* $\boldsymbol{T = [t_1 \ldots t_n]}$ *be a large sting biological sequence and* $\boldsymbol{P = [p_1 \ldots p_m]}$ *be a string pattern where* ***n*** *and* ***m*** *are length of* ***T*** *and* ***P*** *respectively. The process is to locate the exact or approximate occurrences of a string pattern* ***P*** *in* ***T*** *[2] [3].*

For example*, given two disjoint finite sets of alphabets* $\sum$ *and* $\prod$***,*** *the problem is to for all* $t_i \in \sum$ *that exactly or approximately matched a patter* $p \in \prod$*; where* $\sum$ *and* $\prod$ *may be DNA {A, C, G, T}, RNA{A, C, G, U} or Protein {A, C, D, E, F, G, H, K, L, M, N, P, Q, R, S, T, V, W,Y, I} alphabets..*

Bit-parallel string matching is an important technique that is used to analyze biological sequences. It is a technique that simulates non-deterministic finite automata were parallel bit operations are carried out within computer words per clock cycle [4]. It helps improves the performances of string pattern matching process by cutting down the number of these bit operations to a certain level, depending on the computer word size [5] [6]. The use of bit-parallelism approach helps to improve the efficiencies of the early character based pattern matching algorithms like Knuth-Morris-Pratt algorithm (KMP), Boyer-Moore (BM), BMH, BMHS etc. that could not be efficiently applied to biological sequence analysis; this is because of their inability to deal with larger sequences.

Myriad of bit-parallel string pattern matching algorithms were proposed for single or multiple pattern and also for exact and inexact (approximate) pattern matching. In exact pattern matching, we look for the exact occurrences of a given pattern in large body of text while in an approximate pattern matching we look for the occurrences of a pattern with some modification in a large boy of text. For example, after the first proposed method called Shift-OR algorithm, an approximate multi-pattern algorithm was introduced in [20]. An exact single pattern matching algorithm using Backward non-deterministic machine (BNDM) where AND and/or Shift operations are carried out concurrently was introduced by [15]. An advanced method named two- way non-deterministic machine (TNDM) was introduced by [16], unlike in BNDM forward operation is always carried out instead of shifting operation whenever a mismatch occur at last position [12]; [14]. An enhanced exact single pattern matching algorithm that combine wide window and bit parallelism approaches was introduced in [21]. Hybrids approach that reduces the number of false matches using Shift-OR algorithm with Q-gram proposes an approximate multi-patterns matching algorithm that matches one character at a time in [22]. Similarly, [24] proposes using Q-gram and BNDM to implement exact multi-pattern string matching algorithms BNDMq and SBNDMq that read one character at a time at the beginning of each alignment before testing the state variable. An efficient bit-parallel algorithm for alpha and delta matching that combine both dynamic programming and bit-parallelism approaches for music retrieval was proposed in [13]. Although these algorithms have achieved significant improvements on performance, the objective is to have more efficient method that can optimize the performances of the previous methods and be conveniently used to analyze biological data. Therefore, this paper shows how bit-parallelism approach can be used achieve this objective. .

The paper is organized as follows: Introduction was presented in section 1; Section 2 introduces bit-parallel matching for biological sequences while Section 3 discuses parameterized bit-parallel matching for biological sequence. Finally, conclusion is presented in section 4.

2.0 BIT-PARALLEL PATTERN MATCHING FOR BIOLOGICAL SEQUENCE

Bit-parallel pattern matching can be exact or inexact (approximate). Exact matching finds whether the search probability will be either successful or unsuccessful defined as:

Definition2. *Given a pattern **P** of length **n** and a string text **T** of length **m** where **m** >= **n**, we are to find all the exact occurrences of **P** in **T**.*

While inexact pattern matching, sometime referred to as approximate matching or matching with certain number of mismatches can be stated as:

Definition3. *Given a pattern **P** of length **n** and a string text **T** of length **m** such that **m** >= **n**, find all the occurrences of substring **X** in **T** that are similar to **P** with a limited number of say **K** different characters in similar matches.*

Bit-parallel pattern matching can involve single or multiple patterns. Multi-patterns matching uses either of the following scenarios:

(i) Classes of characters, where the text is allowed to match the character of position **j** in any of the pattern. This approach is more suitable for matching with large number of patterns. For example, if we have patterns “ACTG” and “GACT” we can form the supper pattern “{A, G}, {C, A}, {T, C}, {G, T}” that matches the text strings “ACTG”, “AGCT”, “GCTG” and “GACT”.

(ii) Concatenating the 1^{st}, 2^{nd}, 3^{rd} etc. characters of each pattern to form a single pattern. For example, from these patterns “ACTG” and “GACT” we can form a single pattern “AGCATCGT”.

(iii) Concatenating the patterns. For example, from patterns “ACTG” and “GACT” we form a single pattern “ACTGGACT”.

In bit-parallel pattern matching the processes are performed in two phases; pre-processing and matching phases. At the pre-processing phase, bit mask **B[c]** = $\mathbf{b_m \ldots b_1}$. For each symbol **c** in the text is computed as the ith bit of each character is set to **0** if the character appears at position **i** else is set to **1** written from right-to-left and table **B** is then constructed to store these bit masks.

During the search process, the state of the automaton is kept as state vector in a machine word $\mathbf{D = d_m \ldots d_1}$ and transit from state **i** state i+1 when the ith bit of the bit mask **B[C]** of the character **C** is zero. The automaton is then set active at state **i** if and only if the ith bit of the state vector D matches the end of the string read up to the position **i.** The state vector is initially set $\mathbf{D=0^m}$ and for each new character $\mathbf{t_j}$ input, the state vector is updated using the formula $\mathbf{D \leftarrow ((D<<1) \ |0^{m-1} \ 1) \ \&B[t_j]}$, for **m** ≤ **w** where **m** is the pattern length and **w** is the computer word length. A match occurs at position **i** whenever the main significant bit of **D** is **0**.

2.1 Shift-And and Shift-Or algorithms

Shift-And algorithm first build a table for each character of text in a bit mask $\mathbf{b_m \ldots b_1}$, the mask has the ith set if and if **P=C.** the state of the search is kept in a machine word $\mathbf{D = d_m \ldots d_1}$, **d** is set when **p** matches the text end up to current position. Match is reported whenever $\mathbf{d_m}$ is set. Initially the machine word is set to $\mathbf{D=0^m}$ and then for each new

character $\mathbf{t_j}$, **D** is updated using the formula $\mathbf{D \leftarrow ((D\ 1) \mid 0^{m-1}\ 1) \& B[t_j]}$. This algorithm works only when $\mathbf{m \leq n}$ where **m** and **n** are text length and pattern length. Bit parallel shift-And algorithm is a simpler variant of shift-Or with some further advantage of being easily extended to classes of characters i.e. it can be extended to handle wild card character, regular expression approximate search.

In Shift-Or bit mask of table **B** are complemented and operator resulted in an empty string ∅ to the right of **D'** and the suffixes stemming from the empty string is already in **D'.** The operator **|** is used instead of **&** thus the recursive becomes $\mathbf{D' \leftarrow (D\ 1) \mid B[t]}$ and a match is reported when $\mathbf{d_m = \emptyset}$.

There are variant of shift-Or and shift-And algorithms that can be used to handle both parameterized, exact or approximate bit parallel matching with either single or multiple patterns. These include: Parameterized shift-Or and shift-And as in [17] and parameterized shift-Or and shift-And extended with super alphabet as in [18].

Shift-And Algorithm

1 //**Input:** text T of length n and pattern p of length m
2 // **Output:** all occurrences of p in T
3 // Preprocessing:
4 **for** c $\in \sum$ **do** B[c] = 0^m;
5 **for** j $\in$1...m **do** B[p_j] = B[p_j]| $0^{m-j}10^{j-1}$;
6 //Searching:
7 D = 0^m;
8 **for** pos$\in$1… n **do**
9 D = ((D <<1) | 0^{m-1}1& B[t_{pos}];
10 **if** D & $10^{m-1} \neq 0^m$ **then output** "p occurs at position pos - m + 1"

The running time for the searching phase is ***O(n)***, assuming that operations on D can be done in constant time.

Example: *Let Σ = {A, G, C, T} be a set of nucleotide sequence, T= AGCAG be a text string and P=AGCA be a substring pattern.*

B [a] =1001, B[b] = 0010, B [c] = 0100, B[d] = 0000
(1) D = (0000|0001) &1001 =0001 (2) D = (0010|0001) & 0010 =0010
(3) D = (0100|0001) &0100 =0100 (4) D=(1000|0001) &1001=1001, Pattern occurs with shift 0
5) D = (0010|0001) & 0010=0010 (6) D = (0100|0001) & 0100=0100
(7) D = (1000|0001) & 1001=1001 **Pattern occurs with shift 3**
(8) D = (0010|0001) & 0000=0000 (9) D = (0000|0001) & 0010 =0000

2.2 Backward Non-deterministic Matching (BNDM)

It is a variant of the Reverse Factor algorithm that uses bit-parallelism simulation of the suffix automaton of $\mathbf{x^R}$ and the algorithm is only efficient if the pattern length is no longer than the memory-word size of the machine.

The BNDM algorithm uses a table **B** which, for each character **c**, stores a bit mask. The mask in $\mathbf{B_c}$ is set if and only if $\mathbf{x_i = c}$. The search state is kept in a word d = d_{m-1}… d_0, where the pattern length m is less than or equal to the machine word size.

The bit $\mathbf{d_i}$ at iteration **k** is set if an only if x[m - i … m-1-i+k] = y[j + m-k .. j+m-1]. At iteration **0**, **d** is set to $\mathbf{1^{m-1}}$. The formula to update d follows **d'** = (d & B[y_j]) << 1. There is a match if and only if, after iteration **m**, it holds $\mathbf{d_{m-1}=1}$. Whenever $\mathbf{d_{m-1}=1}$, the algorithm has matched a prefix of the pattern in the current window position **j.** The longest prefix matched gives the shift to the next position.

BNMD Algorithm

1. *BNDM ($P=p_1p_2…p_m$, $T=t_1t_2…t_n$)*
2. *for $c\varepsilon\Sigma$ do $B[c]\leftarrow 0^m$*
3. *for $i\varepsilon 1….m$ do $B[P_{m-i+1}]\leftarrow B[P_{m-i+1}] \mid 0^{m-i}\ 1\ 0^{i-1}$*
4. *pos ← 0*
5. *while pos ≤ n - m*
6. *do j ← m, last ← m, $D \leftarrow 1^m$*
7. *while $D \neq 0^m$*
8. *do $D\leftarrow D \& B[t_{pos+j}]$*
9. *j ← j – 1*
10. *if $D \& 10^{m-1} \neq 0^m$*
11. *then if j > 0 then last ←j*
12. *else report an occurrence at pos+1*
13. *D←D<<1*
14. *pos ← pos + last*

For example: *Let T=AGGAGAAGGAAG and P= AAGGAAG, P^r= GAAGGAA over {A, C, G, T}*

B [a] =1100110, B [b] = 0011001, m=7, last=7, j=7, Pos=0

(1) T = [AGGAGAA] GGAAG, D=1111111&1100110 =1100110 j=6, last=6

(2) T = [AGGAGAA] GGAAG, D=1001100 & 1100110 =1000100 j=5, last=5

(3) T = [AGGAGAA] GGAAG, D=0001000 & 0011001=0001000 j=4, last=5

(4) T = [AGGAGAA] GGAAG, D=0010000 & 1100110=0000000 j=3, last=5

We fail to recognize the next **A.** so we shift the window to last. So, we search again in the position: T= AGGAG [AAGGAAG], last=7, j=7, **Pos**=0+5=5

(5) T= AGGAG [AAGGAAG], D=1111111 & 0011001 =0011001 j=6, last=7

(6) T= AGGAG [AAGGAAG], D=0110010 & 1100110 =0100010 j=5, last=7

(7) T= AGGAG [AAGGAAG], D=1000100 & 1100110=1000100 j=4, last=4

(8) T= AGGAG [AAGGAAG], D=0001000 & 0011001=0001000 j=3, last=4

(9) T= AGGAG [AAGGAAG], D=0010000 & 0011001=0010000 j=2, last=4

(10) T= AGGAG [AAGGAAG], D=0100000 & 1100110=0100000 j=1, last=4

(11) T= AGGAG [AAGGAAG], D=1000000 & 1100110=1000000 j=0, last=4

We report a match at **Pos**=6

Other variants of BNDM algorithm are developed for exact or approximate bit parallel matching with either single or multiple patterns, these include: Two way modifications of BNDM (TNDM), Simplified BNDM (SBNDM), BNDM with Q-gram (BNDMq) etc.

2.3 Two way modification of BNDM (TNDM)

This is one of the BNDM variants where the algorithm searches the text string such that the followings hold:

Given an m-length text T string and n-length string pattern P, to compare T and P we consider these three instances:

(i) Sub-string $t_i \in T = P_n$

(ii) Sub-string $t_i \in T$ exist in P_n and $t_i \neq P_n$

(iii) Sub-string $t_i \in T$ does not exist in P_n

In case I and III the algorithm works as in BNDM while in case II, BNDM algorithm searches the text string in the backward direction until either condition III is met or the searching get to the beginning of P while TNDM searches in the forward direction. Because $t_i \neq P_n$ holds in case II, to get the next occurrence the algorithm searches forward instead; resulting in a decrease in the numbers of searched characters. Therefore, TNDM algorithm searches the text string character by character in the forward direction until it get to a character k such that:

(i) Sub-string $\mathbf{t_i \ldots t_k}$ does not exist in $\mathbf{P_n}$ or

(ii) Sub-string $\mathbf{t_i \ldots t_k}$ forms a suffix of $\mathbf{P_n}$

In case I, the shifting can go beyond the previous alignment of P while, in case II, the algorithm continues to search backward from the text position i-1 i.e. reverting back to BNDM algorithm.

Pseodocode

1. *TNDM(P, T)*
2. *Compute table B for each character $c \in \Sigma$ as in BNDM algorithm*
3. *Initiate_shift(P, restore[ ])*
4. *epos ← m*
5. *while epos ≤ n*
6. *do i ← 0, last ← m*
7. *D←B [t_{epos}]*
8. *if ((D&1) = 0 and epos < n) /* D≠B [p_m] */*
9. *then repeat*
10. *i←i+1*
11. *D←D& (B [t_{epos+i}]<< i)*
12. *until D ≠ 0^m and D & 10^i = 0^m and epos + i < n)*
13. *if D=0^m /* already last ←m */*
14. *then goto over*
15. *epos ← epos +i , last ← restore [i]*
16. *repeat /* outside of if of line 7, variation of BNDM */*
17. *i←i+1*
18. *if D & 10^{m-1}≠ 0^m*

19. *then if i<m*
20. *then last ← m-i*
21. *else report an occurrence at epos – m + 1*
22. *goto over*
23. *D ← (D<<1) & B [t_{epos-i}]*
24. *until D ≠ 0^m*
25. *Over:*
26. *epos ← epos + last*

For example: *Let P = ACACA be a string pattern, and T = AAACACAACACA a string text*

1. Pre-processing

We first compute the array restore [ ] as follows:

B[a] = 10101, B[b] = 01010

Step 1: D=11111 & 10101 = 10101, last = 4, ***restore[1]* = 4**

Step 2: D=01010 & 01010 = 01010, last = 4, ***restore[2]* = 4**

Step 3: D=10100 & 10101 = 10100, last = 2, ***restore[3]* = 2**

Step 4: D=01000 & 01010 = 01000, last = 2, ***restore[4]* = 2**

Step 5: D=10000 & 10101 = 10000, last=0, ***restore[5]* = 0**

2. Searching

T = [AAACA]CAACACA

Step 1: epos = 5**,** i=0, last=5

D=B[A] = 10101 = 00000, last = 5-1 = 4, D = 01010 & B [C] = 01010 & 01010 = 01010

Step 2: i=2, D & $10^4 = 0^4$, D= 10100 & 10101= 10100

Step 3: i=3, D & $10^4 \neq 0^4$, last= 5-3 = 2, D = 01000 & 10101, i=1, D & $10^4 \neq 0^4$, **epos = 7**

Step 4: i=0, last=5

D=B[A] =10101, i=1, D & $10^4 \neq 0^4$, last=5-1=4, D=01010 & B[C]=01010 & 01010 = 01010

Step 5: i=2, D & $10^4 = 0^4$, D=10100 & 10101=10100

Step 6: i=3, D & $10^4 \neq 0^4$, **last =5-3=2,** D=01000 & 01010 =01000

Step 7: i=4, D & $10^4 = 0^4$, D= 10000&10101=10000

Step 8: i=5, D & $10^4 \neq 0^4$, Report match at 7-5+1=3, epos=7+2=9

T=AAAC[ACAAC]ACA

Step 9: i=0, last=5, D=B[C] = 01010

Now, D & 1=0 and epos<n, i=1, D=01010 & (10101<<1) = 01010, D & $10^i \neq 0^m$ ⇒b a is a suffix of P, **epos=9+1=10, last=4,** i=2, D=10100 & 10101 =10100, D & $10^4 \neq 0^4$

Last = 5-3=2, i=3, D=01000 & 10101=00000, **epos=10+2=12**

T= AAACACA[ACACA] goto line 15 of the algorithm (BNDM algorithm)

2.4 Simplified BNDM algorithm (SBNDDM)

When there is a match in BNDM or the sub-string $t_h \ldots t_i$ does not appear in P, then there are two possible options for shifting:

(i) If j is the smallest index such that $h < j \leq i$ and $t_j \ldots t_i$ is a prefix of P then the next alignment starts at j

(ii) If $t_j \ldots t_i$ is not a prefix of P, then the next alignment starts at i+1.

While in SBNDM algorithm the shifting is as in BNDM when a match exists. But when the text sub-string $t_h \ldots t_1$ does not appear in the pattern P, then the scanning of prefixes is skipped and the starting position of the next alignment is set to h. This simplifies running the inner loop of the algorithm and resulted in a reduced average shifting length.

Pseudocode

1. *SBNDM(P, T)*
2. *Initialize B and* s_0
3. *Pos* ← *0*
4. *while pos* ≤ *n – m*
5. *do D* ← 1^m, *j* ← *m*
6. *D* ← *(D & B* $[t_{pos+j}]$*)<<1*
7. *J* ← *j-1*
8. *until D=*0^m *or j=0*
9. *if D*≠0^m
10. *then report a match at* ***pos***
11. *pos* ← *pos+*s_0
12. *else pos* ← *pos + j*

For example: *Let P = ACGAC be a string pattern, and T = AAACGACGAC a string text over {A, C, G, T}*

1. Pre-processing

Reversed P= P^r= CAGCA then. B[A]=10010, B[C]=01001, B[G]=00100

2. Searching

T = [AAACG]ACGAC, pos = 0

(1) D = (11111 & 00100) <<1 =01000, j=4, (2) D=(01000 & 01001)<<1 =10000, j=3

(3) D = (10000 & 10010) <<1 =00000, j=2,

(4) T= AA[ACGAC]GAC, pos=0+2=2

D = (11111 & 01001) <<1=10010, j=4, (5) D=(10010 & 10010)<<1=00100, j=3

(6) D = (00100 & 00100) <<1=01000, j=2, (7) D=(01000 & 01001)<<1=10000, j=1

(8) D = (10000 & 10010) <<1=00000, j=0, report match, pos=pos+s_0

(9) T = AAACG[ACGAC], pos=2+3=5

D = (11111 & 01001) <<1 =10010, j=4, (10) D=(10010 & 10010)<<1 =00100, j=3

(11) D = (00100 & 00100) <<1=01000, j=2, (12) D=(01000 & 01001)<<1=10000, j=1

(13) D = (10000 & 10010) <<1 =00000, j=0, report match, pos=pos+s_0, pos=2+3=5

3.0 PARAMETERIZED BIT-PARALLEL PATTERN MATCHING FOR BIOLOGICAL SEQUENCES

The use of pattern matching approach has been a greater tool in solving the problems of bio-sequence analysis. It is applicable in finding homology between fully characterize sequence and newly discovered sequence. However, biological sequences such as that of

RNA and protein have the ability to fold and form some complex structures that are critical to their functions. The complexities of these structures have made the bench mark algorithms unable to efficiently make functional inferences of these biological sequences [25].

3.1 Parameterized Matching

To deal with the complementary base pairing, needed in the formation of structure of biological sequences; parameterized pattern matching method was first proposed in [26]. It is an efficient method that was used in software maintenance where we wish to find equivalency between two codes and was later used as a tool to deal with the complementary behavior of nucleotide that are necessary for forming these complex structures. In parameterized matching, the strings are grouped into two sets, the set of fixed or constant alphabets $\sum_c$ (whose symbols cannot be modified) and that of parameter alphabets $\angle$ (whose symbols can be modified) i.e. two strings **X** and **Y** are said to be parameterized matched if and only if some symbols in **Y** can be transformed or modified to make **Y** equal to **X**. In this type of matching, when looking for occurrences of a pattern **P** in a text string **T**, the symbol of © must match exactly, while the symbols of Π can be renamed. We assume that the pattern P [0…m-1] $\in\Sigma$ and the text T [0…n-1] $\in\Pi$ are string arrays of length **m** and **n** respectively. The pattern P matches the text substring T [j…m-1] if and only if $i \forall i \varepsilon$ {0,1,2…m-1}, it follows that f(P [i] =T[j + i]), where f(.) is any bijective mapping on $\Sigma\cup\Pi$.

***For example**, let Σ = {A, B}, $\angle$ = {X, Y, Z, W} and P=XYABX with parameter matching where f(X↔Z; Y↔W) then P matches the text **T**= ZWABZ.*

To give formal definition of parameterized matching, we first define a parameterized string as:

***Definition 4.** Parameterized string (p-string for short) is a production from two finite set of fixed alphabets $\sum$ and set of parameter alphabets Π such that ($\sum \cap \Pi$) = Ø.*

***For example:** Let $\sum_c$ = {A, B}, Π = {w, x, y, z} be two finite sets of fixed symbols and parameter symbols respectively then strings Q = xyABx, S = zwABz are said to be p-strings.*

So, parameterized matching can be formally defined as:

***Definition 5.** Let $\sum$ and $\angle$ be two disjoint finite set of fixed and parameter alphabets respectively, then two p-strings X and Y over an **m**-length alphabet ($\sum\cup\Pi$) are said to be p-matched if there exist a bijective function say f: $\sum \cup \Pi \leftrightarrow \sum \cup \Pi$ such that X[i] = f(Y[i]) where $1 \leq i \leq$ **m** 1 ≤ i ≤ **m** and in particular **f** established an identity mapping for the symbols Π.*

For parameterized matching for biological sequences, the set of constant or fixed alphabet is empty i.e. $\sum$ = Ø.

The efficiency of the String pattern matching algorithms is greatly affected by the string encoding used. For instance variable width encoding when used can slow down the matching process. In this case the speed can be enhanced by searching the sequence unit instead, though it may result in having false matches if not specifically designed to avoid it [27]. The breakthrough on p-matching by Baker was achieved with introduction of an encoding scheme called the previous encoding (prev-encoding for short) as formally defined:

Definition 9*. Prev-encoding: Given an n-length p-string T and non-negative integer F, the mapping* $(\sum \cup \Pi)^* \rightarrow (\sum \cup F)^*$ *encodes each constant alphabets say* $C \in \sum$ *with the same symbol C and each parameter symbol say* $z \in \Pi$ *to the distance from its previous z, here* $\sum = \emptyset$ *in case of biological sequences as defined in [1].*

$$prev(T_i) = \begin{cases} Ti & if\ Ti \in \sum \\ 0 & if\ Ti, Tj \in \prod and\ Ti \neq Tj \\ i - max\{j | T[i] = T[j], 1 \leq j < i\} & otherwise \end{cases}$$

Therefore, the p-matching problem is to find all the exact p-matches of a pattern in a given text such that *prev*(P) = the *prev*(T) [28] as stated in the lemma below:

Lemma: *Two p-strings S of length* ***n*** *and T of length* ***m*** *are said to be p-matched if prev(S) = prev(T) such that* ***n≤m, n = w****(length of computer word)which can be achieve in* ***O(n)****.*

For example: *Let S=XYABX and T= ZWABZ be two p-strings then prev(S) = 00AB4 and prev(T)=00AB4.*

3.2 Parameterized bit-parallel Shift-And/Or algorithm

The process is similar to that of standard shift-or algorithm but in case, the previous encoding for the pattern and text are first computed at the pre-processing stage**.** At the searching phase, the algorithm considers these encodings for matching in the following manner**.**

Let ***P*** *be a p-string pattern and* ***T*** *a p-string text then* ***P*** *p-matches sub-sting of T at position* ***i*** *if and only if* ***prev(P)*** *is a prefix of* ***p-suffix(T, i)****.*

For parameterized Shift-And set $D=0^m$ originally and, for each new character t_j, update D using the formula $D' \leftarrow ((D<<1)|0^{m-1}1) \& B[t_j]$

Pseudocode

```
1.   P-shift-Or(T, n, P, m)
2.   P'←Encode (P, m)
3.   for i←0 to π -1
4.   do pos[Π[i]] ← -∞
5.   for i←0 to σ + m-1
6.   do B[A[i]] ← ~0
7.   for i←0 to m-1
8.   do B[ P'[i]] ← B[ P'[i]] & ~(1<<i)
9.   for i←1 to m-1
10.   do B[A[σ+i]]← B[A[σ+i]] & (B[A[σ]] | (~0<<i))
11.  D←~0; mm←1<<(m-1)
12.  for i←0 to n-1
13.  do c ←T[i]
14.  if c ∈ ∏
15.   then c ← i-pos[T[i]]
16.   if c>m-1
```

17. then c ← 0
18. pos[T[i]] ← i
19. D←(D<<1)| B[c]
20. if (D & mm) ≠mm
21. then report match at position i- m

For example: *Let P = GAGAG string pattern of length* ***n*** *and T=TATATATA a string of nucleotide of length* ***m*** *over a set of a fixed symbols* Σ = ***Ø*** *and* $\prod$ *= {A, G, C, T}.*

A. Pre-processing phase

Here, the previous encoding for text and pattern are computed as:
Prev(P) = 00222 and prev(T)=00222222

B. Bit mask assignment for the Prev(P)

B[0]=00011, B[1]=B[3]=B[4]=00000 , B[2]=11100
B[1]=00000|(00011&00001)=00000|00001=00001, B[2]=10100|(00001&00011) =10101
B[3]=00000|(11100&00111)=0000|00100=00100, B[4]=00000|(00000&01111) =00000

C. Searching phase

Here, we update D using the formula D' ← ((D<<1)|0^{m-1}1) & B [t_j] and report a match when 1 occurs at the main significant bit of D.

Step 1: Input 0, D = (11111|11110) & 00011=00011, **Step 2**: Input 0, D = (11111|11100) & 00011=00011

Step 3: Input 2, D = (11111|11000) & 10101=10101, **Step 4:** Input 2, D = (11111|10000) & 10101= 10101

Step 5: input 2, D = (11111|00000) & 10101= 10101 Match is reported

Step 6: Input 2, D = (11111|00001) & 10101= 10101, **Step 7:** Input 2, D = (11111|00011) & 10101=01010

Step 8: Input 2, D= (11111|00111) & 10101= 10101 Match is reported

3.3 Parameterized Bit-parallel BNDM Algorithm

The process is the same as that of the standard BNDM algorithm but the use previous string encoding is introduced as in [29]. The encodings for both the strings pattern and text are first computed at the pre-processing stage. The algorithm runs as follows:

Pseudocode

1. PREVP← prev(P) // Find prev encoding of pattern and store in an array
2. RPREVP← prev(REV(P)) // Reverse the array PREVP and store in an array RPREVP
3. for i←0 to σ + m-1
4. do B[A [i]] ←~0>>w –m
5. for i←0 to m-1
6. do B[RPREVP[i]] ← B[RPREVP [i]] &~(1<<i)
7. for i←1 to m-1
8. do B [A [σ+i]]← B [A [σ+i]] & (B[A[σ]]|(~0<<i))
9. mm←1<<(m-1), last←m, j←m, pos←0
10. while pos ≤ n
11. do D←~0

12. k ←pos //need to be consider again
13. for l ←k to m+k-1
14. do find prev of text window of length m and store in temporary array PREVT
15. while D≠0
16. do D←D& B[PREVT[j-1]]
17. j←j-1
18. if D & mm= mm
19. then if j=0
20. then print(" Pattern occurs")
21. else last ←j
22. D←D<<1
23. *pos← pos + last*

For example: *Let P=GAGAG string pattern of length* ***n*** *and T=TATATATA a string of nucleotide of length* ***m*** *over a set of a fixed symbols* Σ = ***Ø*** *and* $\prod$ *= {A, G, C, T}.*

A Pre-processing phase

Here, the previous encoding for text and pattern are computed as:

Prev(P) = 00222, Prev(P^r)=22200 and prev(T)=00222222

B. **Bit mask assignment for the Prev(P^r)**

B[0]=11000, B[1]=B[3]=B[4]=00000 , B[2]=00111

B[1]=00000|(11000&10000)=00000|00001=10000, B[2]=00111|(10000&00011) =00111

B[3]=00000|(00111&00111)=0000|00111=00111, B[4]=00000|(00111&01111) =00111

C. Searching phase

pos=0 , last=5

Step 1: *T = [TATAT]ATA* 00222222

= [00222]ATA, D=11111&11000=01010 j=4

Step 2: T = [00222]ATA, D=11110&11000 =11000 j=3, last=3

Step 3: T = [00222]ATA, D=11100 & 00111=00100 j=2

Step 4: T = [00222]ATA, D=11000&00111 =00000 j=1 , last=1

We shift the window to **pos=pos + last=0+1=1**

Step 10: *T =TAT[ATATA]*

= TAT[22222], D=11111&10101=10101 j=4, last=4

Step 11: TAT[22222], D=01010&01010 = 01010 j=3

Step 12: TAT[22222], D=10100&10101=10100 j=2, last=2

Step 13: TAT[22222], D=01000&01010=01000 j=1

Step 14: TAT[22222], D=10000&10000=10000 j=0, Match Occurs

We shift the window to **pos = pos + last = 3+2 = 5** to look for another match occurrences

Other variant of BNDM algorithms that can be used to handle both parameterized, exact or approximate bit parallel matching with either single or multiple patterns include: Parameterized BNDM, TNDM (Two way modification of BNDM) and Simplified BNDM (SBDM) as in [16] and BNDM with q-gram as in [19] etc.

4. Conclusion

Bit-Parallelism plays a vital role in string comparison and pattern detection which makes it a versatile tool that can be applied in any form of biological sequence analysis such as,

Characterization and/or annotation a newly discovered biological sequences, finding sequence or structural homology between biological sequences etc. Though, it has good application in biological sequence analysis, its application was mostly hampered by the limitation on the pattern length that should be less than or equal to the computer word length [30].

REFERENCES

[1] Richard, B. & Donald, A. (2015). Efficient pattern matching for RNA secondary structure. Theoretical computer science. 592 (2015) 59-71.

[2] Kesavaraj, G., & Sukumaran, S. (2013). A study on classification techniques in data mining. In Computing, Communications and Networking Technologies (ICCCNT), Fourth International Conference on (pp. 1-7). IEEE.

[3] Zhong, N., Li, Y., & Wu, S. T. (2012). Effective pattern discovery for text mining. IEEE transactions on knowledge and data engineering, 24(1), 30-44.

[4] Gupta, S. &Rasool, A. (2014). Bit-parallel string matching algorithms: A survey. International journal of computer application. (0975-8887). Volume 95 No. 10.

[5] Cantone, D., & Faro, S. (2014). Efficient Online Abelian Pattern Matching in Strings by Simulating Reactive Multi- Automata. In Stringology (pp. 30-42).

[6] Faro, S., & Lecroq, T. (2012). Twenty years of bit-parallelism in string matching. Festschrift for Borivoj Melichar, 72-101.

[7] Allali, J., Sagot, M. (2005). A new distance for high level RNA secondary structure comparison, EEE/ACM Trans. Computational Biol. Bioinformatics. 2(1) (2005) 3–14.

[8] Xu, Y., Wang, L., Zhao, H. & Li, J. (2005). Exact matching of RNA secondary structure patterns, Theoretical computer Sci. 335(1) (2005) 53–66.

[9] Heyne, S., Will, S., Beckstette, M. & Backofen, R. (2009). Lightweight comparison of RNAs based on exact sequence-structure matches, Bioinformatics 25(16) (2009) 2095–2102.

[10] Mauri, G. & Pavesi, G. (2005). Algorithms for pattern matching and discovery in RNA secondary structure, Theoretical computer Sci. 335(1) (2005) 29–51.

[11] Strothmann, D. (2007). The affix array data structure and its applications to RNA secondary structure analysis, Theoretical computer Sci. 389(1–2) (2007) 278–294.

[12] Ďurian, B., Peltola, H., Salmela, L., & Tarhio, J. (2010). Bit-parallel search algorithms for long patterns. In International Symposium on Experimental Algorithms (pp. 129-140). Springer, Berlin, Heidelberg.

[13] Grabowski, S., & Fredriksson, K. (2008). Bit-parallel string matching under hamming distance in O (n⌈ m/w⌉) worst case time. *Information Processing Letters*, *105*(5), 182-187.

[14] Prasad, R., Sharma, A. K., Singh, A., Agarwal, S., & Misra, S. (2011). Efficient bit-parallel multi-patterns approximate string matching algorithms. *Scientific Research and Essays*, *6*(4), 876-881.

[15] Navarro, G., & Raffinot, M. (1998). A bit-parallel approach to suffix automata: Fast extended string matching. In *Annual Symposium on Combinatorial Pattern Matching* (pp. 14-33). Springer, Berlin, Heidelberg.

[16] Pettola, H. & Tarhio, J. (2003). Alternative algorithm for bit-parallel string matching. 10th international symposium SPIRE 2003, Manaus, Brazil. DOI: 10.1007/978-3-540-39984.

[17] Prasad, R. & Agarwal S. (2008). Parameterized shift-And string matching algorithm using super alphabet. International conference in computing and engineering (ECCCE, 2008).

[18] Krishna, K., Prasad, R. & Agarwal S. (2009). Multi-pattern Parameterized shift-And string matching algorithm. DOI:10.1145/1764810.1764822.

[19] Prasad, R. & Agarwal S. (2010). Software maintenance by multi-pattern parameterized string matching with q-gram. ACM SIGSOFT software and engineering notes. 35(3), 1-5.

[20] Ricardo A. Baeza-Yates and Gaston H. Gonnet,"A New Approach to Text Searching", In Communications of the ACM, pp. 74-82, Oct 1992.

[21] Longtao Hea, Binxing Fangaand Jie Sui," The wide window string matching algorithm" In the proceedings of Theoretical Computer Science of Elsevier, Vol. 332, pp. 391-404, 2005.

[22] L. Salmela, J. Tarhio, and J. Kytojoki, "Multi pattern string matching with q-grams", Journal of Experimental Algorithms, Volume 11, pp. 1-19, 2006.

[23] Branislav Durian, Jan Holub, Hannu Peltola and Jarma Tarhio,"Tuning BNDM with q-grams", In the proc. Of workshop on algorithm engineering and experiments, SIAM USA, pp. 29-37, 2009.

[24] Changsheng Miao, Guiran Chang and Xingwei Wang," Filtering Based Multiple String Matching Algorithm Combining q-Grams and BNDM", In proc. Of Fourth International Conference on Genetic and Evolutionary Computing, 2010.

[25 Jeff Anderson-Lee et al., (2016). Principles for Predicting RNA Secondary Structure Design Difficulty. *J. Mol Biol*, *428*, 748–757. https://doi.org/org/10.1016/j.jmb.2015.11.013

[26] Baker, B. (1993). A theory of parameterized pattern matching: algorithms and applications. *In: Proceedings of STOC '93, ACM, New York*, 71–80.

[27] J. Mendivelso, et al., A brief history of parameterized matching problems, Discrete Applied Mathematics (2018), https://doi.org/10.1016/j.dam.2018.07.017.

[28] Islam, T., & Talukder, K. H. (2017). An improved algorithm for string matching using index based shifting approach. In *Computing, Analytics and Security Trends (CAST), International Conference of Computer and Information Technology (ICCIT),* (pp. 138-143). IEEE.

[29] Prasad, R. (2016). Efficient Parameterized Word Matching Using Bit-Parallelism and Partitioning the Text. *International Research Journal of Electronics and Computer Engineering*, *2*(2), 20-24.

[30] Prasad, R., Sharma, A. K., Singh, A., Agarwal, S., & Misra, S. (2011). Efficient bit-parallel multi-patterns approximate string matching algorithms. *Scientific Research and Essays*, *6*(4), 876-881.