***Abstract***

DNA data provide us a considerable amount of information regarding our biological data, necessary to study ourselves and learn about variant characteristics. Even being able to extract the DNA from cells and sequence it, there is a long way to process it in one step.

Over past years, biologists evolved attempting to “decipher” the DNA code. Keyword search and string matching algorithms play a vital role in computational biology. Relationships between sequences define the biological functional and structural of the biological sequences. Finding such similarities is a challenging research area, comprehending BigData, that can bring a better understanding of the evolutionary and genetic relationships among the genes. This paper studied and analyzed different kinds of string matching algorithms used for biological sequencing, and their complexity and performance are assessed.

***Introduction***

Keyword search and matching are techniques to discover patterns inside specific strings. Algorithms for matching, are used to discover matches between patterns and input strings. For instance, V represents an alphabet; in V there are characters or symbols. Assuming, V={A,G} , then AAGAG is a string. Patterns are labeled by P(1...M) , while the string is labeled as T(1...N) . Pattern can occur inside a string by using a shifting operation.

Text-editing applications can also benefit from the aid of string matching algorithms, aiding and improving responsiveness while writing. There are two main approaches for string matching, first is exact matching, for instance: Smith-Waterman (SW); Needleman Wunsch (NW); Boyer Moore Horspool (BMH); Dynamic Programming; Knuth Morris Pratt (KMP). Second approach, is approximate matching, also known by Fuzzy string searching, for instance: Rabin Karp; Brute Force.

Many algorithms try to give solutions for the string matching problems like, pattern matching using wide window, approximate matching, polymorphic matching, minimize mismatches, prefix/suffix matching, similarity measure, longest commons sub-sequence (using dynamic programming algorithms), BHM, Brute Force, KMP, Quick search, Rabin Karp (Singla and Garg 2012).

***Related work***

***Boyer Moore Algorithm***

Boyer Moore algorithm (BM) for string search and match is a standard benchmark algorithm, considered one of the most efficient when the alphabet comprises a small size of characters, used on standard editors to perform string search. SO this algorithm is often used in bioinformatics for disease detection. This algorithm works right to left by matching two sequences, thins method consists on a backward approach. When there is any mismatch it means that the patter was found, otherwise the sequence is moved to right (shifted) and a new match attempt is performed (Martin et al. 2005). The pseudo code in Listing 1.2 illustrates how the Boyer Moore algorithm works. Where as input, the text T of size n and pattern P of size m, and as output, the first index of the string T equal to P or −1 if no match is found.

***Rabin Karp Algorithm***

Rabin Karp (RK) algorithm is used to look for similarities in two sequences, proteins, i.e. a high sequence similarity means significant structural or functional similarity. RK utilizes a hash function to make the string searching faster. The hash value for the pattern is calculated, used then to compare with sub-sequences of the text. When the hash values are different, RK algorithm estimates the hash for the next matching sequence of characters. If hash values are equal, RK algorithm uses brute-force to compare sequence pattern with the text. RK algorithm efficiency is based on the computation of hash values of text sub-strings.

Rolling hash functions analyze sub-strings as a number, the base of this functions, usually is a large prime number. For instance, considering the sub-string “AC” and the base 1011, the hash would be 65×10111+67×10110=65782 , where “A” is 65 and “C” is 67 in the ASCII table.

Lets consider a M-character sequence with M-digit number, base b, where b is the number of letters in the alphabet, the text mapped into sub-sequence t[1...i+M−1] .

(8) X(i)=t[i]×bM−1+t[i+1]×bM−2+...+t[i+M−1] //Equation 8 is used to compute sub-sequences of specific sequences.

Given x(i) it is possible to compute x(i+1) for the next sub-sequence t[i+1...i+M] as in Eq. 9

(9) X(i+1)=t [i+1]×bM−1+t[i+2]×bM−2+...+t[i+M]//Equation 9 shows how to discover the next sub-sequence for the predecessor.

(10) h(i)=((t[i]×bM−1modq)+(t[i+1]×bM−2modq)+…+(t [i+M−1]modq))//Equation 10 is used to calculate hash values for the sub sequences. Where:

x(i) represents the text sub-sequence t[i] ; h(i) represents the hash function; b represents the base of the string; q is the prime number;

Algorithms like Knuth Morris Pratt are easier to implement, because they never need to back-reanalyze in the sequence, however, requires more space. Rabin Karp algorithm requires extra space for matching. Brute Force, has the advantage of not needing any pre-processing. However, it is slow. AhoCorasick algorithm is commonly used for a multi-pattern string match. CommentZ-Walter is slower to reach a result, Boyer More is faster when using large sequences, avoiding many comparisons. Boyer More in best case scenario complexity is sub-linear. As future work, it is proposed a parallel algorithm for fuzzy string matching, using artificial intelligence neural networks for better performance and accuracy.