Summary of paper

Abstract :

The reason for development methods in bioinformatics is the increase of biological. Pattern matching is very important operation in different computational stages. Pattern matching helps to find the places of similar subsequences in a database, it checks all database or text to determined places of the pattern in the text. The paper introduces ways for pattern matching to speed up searching for DNA sequences. The suggested algorithm helps to improve performance by using Keywords during search to save time.

Introduction :

- The operation of suggested algorithms divided into before processing and matching phase.
- The paper focus on solving the problem of accurate pattern matching which find all occurrence in the text. the paper introduces three algorithms.
- in preprocessing phase, possible time intervals for the text are recognized to be matched with the pattern, this intervals called windows.
- in matching phase, windows are scanned carefully to match with the pattern. the fewer windows in the preprocessing phase, the less time it takes to check windows in the match phase.
- The first proposed algorithm aims to determined windows by discovering first and last letter in the text and look at the same time to first and last letter in the pattern.
- The second algorithm makes comparisons depending on words, the words are processed using processor power. the computational length of processors almost 32 or 64bits in each execution cycle. it can process 4 or 8 bytes of data, it can compare 4 or 8 letters with 4 or 8 other letters.
- The third algorithm focus on word of the pattern which has the fewest number of iterations in the text and this will reduce the number of windows detected.

Related Work :

▼ Brute Force(BF) is a primary method that preprocesses neither the text nor the pattern. BF carries out by sliding the pattern over the text one by one from left to right. The disadvantage of BF is the high of consumption of time.

▼ There are methods based on Deterministic Finite Automata(DFA), that combine the dynamic programming approach and DFA. Because of the use of a finite automaton, this type of methods is not scalable for large sequences. In addition, dynamic programming leads to higher memory requirements

▼ KMP algorithm preprocesses pattern size pat[] and constructs an auxiliary LPS[] (longest proper prefix or suffix) of size n (same size of pattern) which is used to skip characters while matching.

▼ Boyer-Moore algorithm does preprocessing over the pattern so that the pattern can be shifted by more than one. this algorithm first matches the pattern's last character. At the end of matching phase, it computes the shift increment. if mismatch occurs, to decrease the number of comparison. two useful rules (bad character and good suffix) are utilized. The disadvantage of this algorithm is the dependency of its preprocessing time on the pattern length and alphabet size.

▼ Divide and Conquer Pattern Matching(DCPM):the text is scanned for the rightmost character of the pattern. to detect the leftmost character of the pattern, the text is scanned again. DCPM requires two passes of the text and some computations to determine the windows. In the matching phase, the algorithm checks the other characters of the windows. if all characters of the pattern and the windows of the text are matched, then complete sameness occurs.

Methodology :

Algorithm 1 preprocessing phase of FLPM algorithm

count=0 , number-window=0

while count ≤n-m , Do

if t[count]=p[0] , then

if t[count+m-1]=p[m-1] , then

window-index[number-window]=count

number-window=number-window+1

End if

End if , count=count+1

End while.

**Algorithm 2 Matching phase of FLPM algorithm

count=0 , number-match=0

while count < number-window , Do

S=window-index[count] , C=1

while C ≤ m-2 , Do

if p[C] not equal t[S+C] , then

break /*Exit the current loop */

End if , c=c+1

End while

if C=m-1 , then

match-index [number-match] = S

number-match =number match+1

end if , count = count +1

end while.

Results :

This section compares performance of algorithms(FLPM,PAPM and LFPM) with Brute Force(BF),Boyer-Moore(BM) and Divide and Conquer Pattern Matching(DCPM)algorithms.

A- The time of preprocessing phase:

-In FLPM one pass across the text to discover first and last character of pattern.

-DCPM has two passes: for the leftmost character and another for the rightmost.

-PAPM searches for the first word of the pattern in the text.

FLPM finds the least frequent word consumes long time. PAPM and LFPM consumes the least amount of time among the simulated algorithms.

B- The Time of Matching Phase: presents the time cost of matching phase for algorithms. DCPM compares each element in the rightmost table to all elements, so it consuming very time. PAPM and LFPM use word processing to match the windows and the pattern is much faster than the simulated character-based algorithms, BF,DCPM,BM and FLPM.

C-Total Time: provides the sum of time costs. PAPM and LFPM reduce the time to execute pattern matching. In LFPM, there is a difference between the repetition number of the least frequent word and those of other words of the pattern. The higher value of this difference leads to more improvement in LFPM performance.