Simple and Efficient Pattern Matching Algorithms for Biological Sequences

Dep: Medical Informatics

Name: Nada Mohamed El Sayed EL Mahdy
PhD: Sara El Metwally

Abstract:

- The increase in the size of biological data and the need to search for patterns.
- The need to decrease the running time of the algorithm.
- The paper presented three pattern matching algorithms for decreasing the time of the finding pattern in large DNA sequences.
- Used word processing instead of (character processing used in other algorithms) and searching for the least frequent word of the pattern in the sequence.

Introduction:

- Pattern matching is looking for all positions of appearance of the pattern in the text.
- The problem with pattern matching and its applications in important fields such as text processing, search engines, question answer systems, and especially, in different scopes in computational bioinformatics is increasing, such as
  - finding the positions of pattern(s) which may be remarkable for amino acid(s) or specific gene(s) or detecting disease.
  - Sequences Comparison to detect the similarity between the sequences.
- The problem is the increasing size of biological data and many existing algorithms are not suitable for large DNA sequences due to a long running time.
- The paper deals with exact pattern matching and proposes three algorithms to reduce the drawbacks of the previous algorithms.
- The operation of the algorithms is divided into:
  - The preprocessing phase: the potential intervals of the text to be matched with the pattern are called candidate windows.
  - The matching phase: compare candidate windows with the pattern.
  - And aim to decrease the number of the windows.
- The first algorithm for finding windows by searching the text for the first and the last character of the pattern at the same time instead of separately as in some other algorithms.
- Due processors can handle 4 or 8 bytes of data (4 or 8 characters) and are referred to as a word processor which can compared to another word.
  The second algorithm for word-based comparisons by processing the word using the benefit of the processing power of the processor instead of character-based algorithms to decrease the running time and enhance the performance.
- The third algorithm looks for the least frequently occurring word of the pattern in the text .

Related work:

- **Brute Force (BF):** It is a basic method that doesn't perform any processing on pattern or text, only comparing the text and the pattern character by character from the left side and shifting the pattern by one position in the next comparison if a match or mismatch occurs. The drawback is the long running time of BF.
- **Deterministic Finite Automata (DFA):** There are methods that rely on DFA and the dynamic programming approach. Finite automation led to it being not

often suitable for large sequences. Need a large size of memory as a result of dynamic programming.

- **Knuth Morris Pratt Algorithm (KMP):** It makes a comparison between the text and the pattern from the left side. Make preprocessing on the pattern and creating the longest proper prefix which is also suffix (LPS) table of the same size as the pattern. LPS is used to shift the pattern while matching. KMP works well even if the size of the alphabet is large. When the alphabet size or pattern length is small, KMP takes a long time to run.
- **Boyer Moore Algorithm (BM):** It makes a comparison between the text and the last character of the pattern. Perform preprocessing over the pattern and creating two tables by using a bad character rule and a good suffix rule to compute the number of pattern shifts and take the maximum number of them to reduce comparisons when a mismatch occurs. The drawback is it takes preprocessing time to create two tables relying on the length of the pattern and alphabet size.
- **The Divide and Conquer Pattern Matching (DCPM):** It is a comparison-based algorithm. At the start, the preprocessing phase searches in the text for the last character and the first character of the pattern separately and saved the indexes of the findings in the rightmost character table and the leftmost character table respectively. By using the two tables, find the windows of the text by computing the distance between the first character and the last character of the window is the same as the length of the pattern. In the matching phase, compare the other characters of the windows with the pattern. The first algorithm of the current paper promotes the DCPM by determining the windows with the one search in the text.

Methods:

***(pattern" p" of length m, text" t" of length n, s: position of character in text "*0 <= s <= n- m", s_i: start index of window*, word_len: word length "the size of the register of the processor" , REF_len :the length of the reference array ,HRG: human reference genome)*

I. **First-Last Pattern Matching Algorithm (FLPM):** FLPM is an improvement of DCPM and uses a character-based pattern matching algorithm.
   - The Preprocessing Phase: The text is searched to find the candidate windows of length m. Search for the first character of pattern p [0] in interval t [0… n-m] if p [0] =t[s] immediately, the algorithm checks the similarity of the last character of pattern p[m-1] to t[s+m-1] to determine the end of the window if p[m-1] =t[s+m-1] then the t [s… s+m-1] is a candidate window to be checked in the matching phase. This algorithm focuses on the first and the last character of the pattern. It may be considered characters in other positions m/4, m/2 and 3m/4.
   - The Matching phase: Make a comparison between the candidate windows t [s_i+1…s_i+m-2] and the pattern p [1…m-2], consider the first and the last character of the pattern and the window shouldn't be compared again as they are checked in preprocessing phase. If all characters are matched, then the pattern is found. After a match or mismatch, this phase continues by checking the next candidate window.

II. **Processor-Aware Pattern Matching Algorithm (PAPM):** PAPM is applied to DNA sequences only. PAPM uses a word-based pattern matching algorithm, by taking the advantage of the processing power of the processor.
   - The Preprocessing Phase: searching in the text t [0...n-m] to find the first word of p [0...*word_len*-1] of the windows "have the same length of the pattern" and save the start indexes in *window_index* array. The more characters of the word, the smaller number of windows, which may decrease the time in the matching phase.
   - The Matching Phase: Make comparison between the words of pattern and the words of the candidate windows, consider the matching start at the second words of them. The start position to match has the index equaling the pattern length subtracted from the word length. The algorithm is performed correctly, even if the length of the pattern and windows are not an integer multiple of the word length.

III. **Least Frequency Pattern Matching Algorithm (LFPM):** LFPM is an enhancement of PAPM and that is applied to any type of sequences. LFPM is a good choice for searching for many patterns but not suitable for few patterns due to time overhead. To reduce the number of windows and the running time of the algorithm, it looks for the word of the pattern with the fewest occurrences in the text.
   - In the first step before Preprocessing Phase: Create the *fre_table that* Compute the number of occurrences of different words of the pattern in the text. The number of all possible words Depend on the number of the alphabet of the text as the number of the alphabet of DNA= {A, G, C, T} is 4 and the size of the computer's register=*word_len*, then it is computed by 4^{word_len} . The table has 4^{word_len} rows and *word_len* + 1 column. Each row has the word's characters from column 0 to *word_len*-1. In the last column *"word_len* column*"* the frequency of the word in the text. Because the size of memory required to hold the *freq_table* is reasonable, it can be stored in the main memory. The taken time to fill the table is θ(*word_len x (REF_len – word_len* +1)). Although filling a table is time-consuming, it is done once on the text "Reference" and can use it for searching every pattern over the alphabet. If the reference is modified over time, the table can be periodically updated. The size of the table increases as the size of the sequence data increases.
   - The Preprocessing Phase:
     - at the beginning of this phase: Choose the word of the pattern that has the lowest frequency using *freq_table* and save the star index of it for the comparison. the taken time to determinate the least frequency word is θ (*word_len* x *(m – word_len))*. However, as *word_len* is fixed for each computer, the time may be stated as θ(m)
     - in the second step in this phase: Similar to PAPM, searching in the text to find the windows by finding the least frequency word, consider the

star position of the chosen word in the windows equal to the number of the skip words. The number of windows using LFPM is often less than in the previous algorithms

- The Matching Phase: Make comparison between the words of pattern and the words of the candidate windows

Results:

The specifications of the computing environment for executing different simulated algorithms were as follows: Intel Rcore™2 Duo CPU T6600 (a 2.2 GHz clock), A 2GB Memory, Acer (Aspire 5738), Windows 7 ultimate 32 bit. Use c programming language.

- Due to 32-bit computer, in PAPM and LFPM the *word_len* is 4 bytes *(4 characters)*.
- Use HRG as a reference to all algorithms and in LFPM to create the *freq_table*.
- The overhead time for creating the table is fixed at 12 milliseconds and was ignored in the calculations because it was only created once.
- Ten pattern were searched in the HRG for all algorithms.

A. The Time of Preprocessing Phase: the time of the preprocessing phase for different algorithms over the pattern length. The BM's time is very small to only process the pattern. In FLPM, takes one pass through the text to look for the first and last character of the pattern concurrently. In contrast, DCPM takes two passes to look for the first and last character of the pattern separately. PAPM takes one pass to look for the first word of the pattern in the text. It is expected PAPM finds fewer windows than FLPM.LFPM searches for the least frequency word of the pattern using *freq_table ,* which takes time and it finds fewer windows. LFPM and PAPM take less time than other algorithms.BF doesn't have preprocessing phase.

B. The Time of Matching Phase: FLPM, PAPM and LFPM are better than other algorithms due to determining the low number of windows in the preprocessing phase. PAPM and LFMP are faster than character-based algorithms "BF, DCPM, BM and FLPM" in matching the windows with the pattern.

C. Total Time: The sum of the time costs for the two phases. Using word processing by PAPM and LFPM can reduce the required time. 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. In terms of time cost, LFPM outperforms the other algorithms due to decreasing the number of the windows.