GENOMICS PAPER

Simple and Efficient Pattern Matching Algorithms for Biological Sequences

Zeyad elsayed elkomy rashed

Abstract:

The growth of biological data is pushing us to discover solutions in many areas of computational bioinformatics.In different stages of the computational pipelines,pattern matching is a very practical operation.For example,pattern matching enables us to find the locations of particular DNA sequences or subsequence in a database.Furthermore, in these expanding biological databases, some patterns are updated over time. To perform faster searches, high-speed pattern matching algorithms are needed. In paper we introduce pattern matching algorithms that are used to speed up search on large DNAsequences.The proposed algorithms raise performance by utilizing word processing (instead of the character processing) and also by searching the least frequent word of the pattern in the sequence.The experimental results showed that these algorithms are distinguished over other algorithms in terms of Speed and accuracy .

INTRODUCTION:

In the pattern matching problem, a sequence is scanned to detect the locations of it. It is important to address these problems due to their applications in various fields such as image and text processing, search engines, and information retrieval.Notably, the pattern matching problem arises in different scopes of computational bioinformatics,example biomarker discovery, sequence alignment.In these disciplines, there is a need to recognize the locations of multiple patterns, including those of amino acids and nucleotides in Databases and we use it in different fields. Although these algorithms are efficient the development of it is still required. because many current algorithms may not be well scalable for databases or large DNA sequences due to high-computational costs.the current paper focuses on the exact pattern matching problem which finds all the occurrences of a pattern in a text by using algorithms to Repair previous defects and it is divided into a Different stages.Nowadays, the computational length of almost all processors is 32 or 64 bits in each execution cycle so they can process 4 or 8 bytes of data instantly. Therefore, 4 or 8 characters, indicated by a word.we introduces also an algorithms to conduct word-based comparisons. The word processing is performed by utilizing power of the processor. This approach creates a new class of string-matching algorithms that improve the performance of character-based algorithms. As a result, the performance improves in terms of time cost.

Related work:

In the pattern matching algorithm, Brute Force (BF) is a primary method which preprocesses neither the text nor the pattern BF Comparing character by character from left to right. After either a match or mismatch, it is shifted one position to the right and the matching is restarted from the first character of the pattern.but The disadvantage of BF is that it consumes a lot of time.There are methods based on Deterministic Finite Automata(DFA) that combine the dynamic programming approach and DFA. Due to the use of a finite automaton, these methods are not often scalable for large sequences. In addition, the memory requirements are greater because of the usage of dynamic programming.the KMP algorithm which performs the comparison from the left side. If a mismatch happens, it moves to the right by holding the longest overlap of a suffix of the matched text and a prefix of the pattern. Although it performs well when the alphabet size is large, the KMP algorithm requires a long run time when either the alphabet size is small or the length of the pattern is short.The Boyer-Moore first matches the pattern's last character. At the end of the matching phase, it computes the shift increment.To decrease the number of comparisons when a mismatch occurs, two useful rules (bad character and good suffix) are utilized. The disadvantage of the Boyer-Moore algorithm is the dependency of its preprocessing time on the pattern length and alphabet size.

FIRST- LAST PATTERN MATCHING ALGORITHM

Algorithm 1 Preprocessing Phase of FLPM Algorithm

Input: Text *t* and pattern *p* stored in the arrays of *t*[0..*nn*1] and *p*[0..*m m* 1], respectively, over fifinite alphabet. **Output:** The number of windows identifified in this phase and their start indexes.

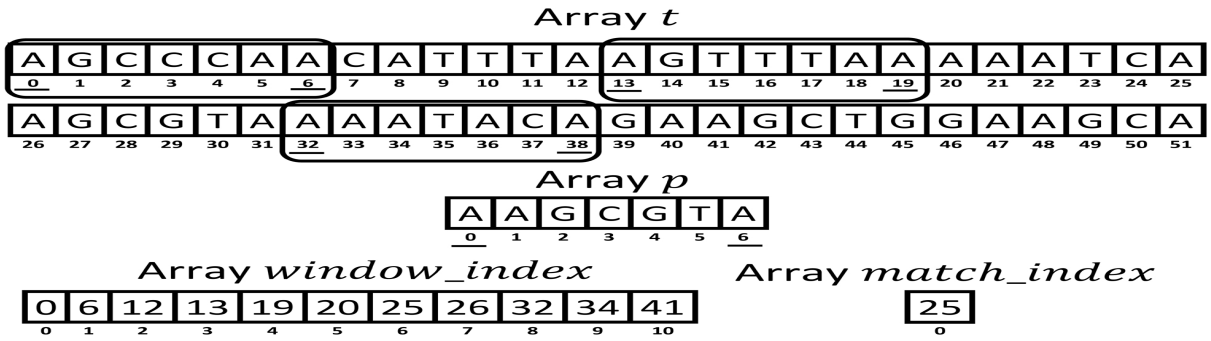
1. *count* ← 0 2. *num_window* ← 0 3. **WHILE** *count* ≤ *n n m* **DO** 4. **IF** *t* [*count*] = *p*[0] **THEN**

5. **IF** *t* [*count* + *m m* 1] = *p*[*m m* 1] **THEN** 6. *window_index*[*num_window*] ← *count*
7. *num_window* ← *num_window* + 1 8. **END-IF**
9. **END-IF** 10. *count* ← *count* + 1 11. **END-WHILE**

Algorithm 2 Matching Phase of FLPM Algorithm

Input: The number of windows identifified in the preprocessing phase and their start indexes.
Output: The start index for all occurrences of pattern *p* in text *t*.

1. *count* ← 0 2. *num_match* ← 3. **WHILE** *count* < *num_window* **DO**
4. *s* ← *window_index*[*count*] 5. *c* ← 1 6. **WHILE** *c* ≤ *m m* 2 **DO**
7. **IF** *p* [*c*] 6= *t*[*s* + *c*] **THEN** 8. BREAK /∗ Exit the current loop∗ / 9. **END-IF**
10. *c* ← *c* + 1 11. **END-WHILE** 12. **IF** *c* = *m m* 1 **THEN**
13. *match_index*[*num_match*] ← *s* 14. *num_match* ← *num_match* + 1 15. **END-IF**
16. *count* ← *count* + 1 17. **END-WHILE**



CONCLUSION

The current paper introduces three new algorithms: FLPM, PAPM, and LFPM. Similar to previous works, LFPM is a character-based pattern matching algorithm, while PAPM and LFPM perform based on the word processing approach. Furthermore, LFPM searches for the lowest frequency word of the pattern in order to minimize the algorithm run time. The present work's experimental results reveal that the proposed algorithms, especially LFPM, override the other simulated algorithms in terms of time cost. This improvement is mainly due to having decreased the number of found windows. Moreover, as this paper provides solutions for exact pattern matching, future research may investigate the algorithms supporting approximate matching.

REFERENCES

[1] P. Montanari, I. Bartolini, P. Ciaccia, M. Patella, S. Ceri, and M. Masseroli, ''Pattern similarity search in genomic sequences,'' *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 3053–3067,2016.

[2] V. Abrishami, A. Zaldívar-Peraza, J. M. de la Rosa-Trevín, J. Vargas, J. Otón, R. Marabini, Y. Shkolnisky, J. M. Carazo, and C. O. S. Sorzano, ''A pattern matching approach to the automatic selection of particles from low-contrast electron micrographs,'' *Bioinformatics*, vol. 29, no. 19, pp. 2460–2468, Oct. 2013.

[3] S. Faro and T. Lecroq, ''The exact online string matching problem,'' *CSURACM Comput. Surv.*, vol. 45, no. 2, pp. 1–42, Feb. 2013.

[4] M. Tahir, M. Sardaraz, and A. A. Ikram, ''EPMA: Effificient pattern matching algorithm for DNA sequences,'' *Expert Syst. Appl.*, vol. 80, pp. 162–170, Sep. 2017.

[5] S. I. Hakak, A. Kamsin, P. Shivakumara, G. A. Gilkar, W. Z. Khan, and M. Imran, ''Exact string matching algorithms: Survey, issues, and future research directions,'' *IEEE Access*, vol. 7, pp. 69614–69637, 2019.

[6] M. Sazvar, M. Naghibzadeh, and N. Saadati, ''Quick-MLCS: A new algorithm for the multiple longest common subsequence problem,'' in *Proc. 5th Int. Conf. Comput. Sci. Softw. Eng. (CSE)*, 2012, pp. 61–66.

[7] V. Y. Gudur and A. Acharyya, ''Hardware-software codesign based accelerated and reconfifigurable methodology for string matching in computational bioinformatics applications,'' *IEEE/ACM Trans. Comput. Biol. Bioinf.*, to be published.