DOUBLE HASHING APPROACH FOR PATTERN MATCHING OF DNA SEQUENCES

Md. Rafi Alam [#], Sumaya Jahan [#], Md. Mazharul Islam [#]

[#] Dept. of Computer Science & Engineering
[#]Rajshahi University of Engineering & Technology, Rajshahi, Bangladesh

ABSTRACT

Pattern matching in DNA sequences has been an interesting as well as challenging topic for years in bio-informatics. Since there are four bases (A,C,T and G) to represent a DNA sequence, the pattern matching between DNA sequences is basically a string matching problem. Hashing is a well known technique for string matching problems. But collision can occur in case of large DNA sequences. This paper evaluates an extension of hashing technique which is called Double Hashing that minimizes the collision problem. Although this technique takes 0.79 seconds longer on average to execute than hashing according to the cited dataset, it provides 99.89% higher accuracy in reducing collision issues.

Index Terms— DNA sequence, Hashing, Double Hashing, Collision, Accuracy

1. INTRODUCTION

Pattern matching of DNA sequences is a subject of great importance in Bio-informatics and Molecular Biology as DNA is a heredity material in all the organisms. DNA pattern searches are employed in a variety of sectors including disease diagnosis, agriculture, forensic science etc. In computer science, a DNA sequence is mainly an array of character(s) known as a string, and there are a variety of string matching techniques that can be implemented to search DNA patterns efficiently.

Based on the number of patterns searched at a particular time, pattern matching technique can be divided into two categories [1].

- Single pattern matching
- Multiple pattern matching

Here, in single pattern matching, at most one string is searched at a time whereas in multiple pattern matching more than one pattern can be searched. Pattern matching can also be categorized as:

- approximate/inexact pattern matching
- exact pattern matching

In approximate pattern matching, errors can be significant up to a certain level but in exact pattern matching, the pattern should be fully matched with any substring from the main DNA sequence.

The simplest string matching algorithm is naive approach [2] which don't preprocess the main text or the pattern. The running time complexity of this approach is *O(mn)* [3]. As the DNA sequences are most of the time a large string, some efficient methods were introduced where the string was preprocessed. The Knuth–Morris–Pratt (KMP) [4] algorithm is a prefix approach where degenerating property is used. In this approach, comparisons begin from left side and it requires $O(m)$ time for preprocessing and $O(n + m)$ for searching [5]. Boyer-Moore (BM) algorithm [6, 7] is based on right to left scan, bad character rule and good suffix shift rule [8]. The Divide and Conquer Pattern Matching (DCPM) [5] is an algorithm based on comparison. In the preprocessing phase, comparison starts from the character of right side from the pattern and findings are stored in rightmost table. The text is scanned again to detect the leftmost character of the pattern and findings are stored in leftmost table. Then, any match in the value of the table is found out in the exact pattern matching technique. Prefix tree/trie [9] is formed of nodes and arcs. Arc labels contain the characters of the text string. In case of searching the pattern, it is necessary to go through the preferred nodes sequentially as well as check whether it is an end point label and there the pattern also ends or not.

Hashing [10] is also a frequently used algorithm where a key is transformed into another value called hash value using some user specified hash function. This precalculation leads to find a substring within the main DNA sequence at *O(1)* running time complexity. However, based on the constraints and hash function there is a higher probability that several hash values may collide. This collision probability of hashing hampers the accuracy of DNA pattern matching.

In this paper, we have proposed a technique called Double Hashing for pattern matching of the DNA sequences where probability of collision(s) is highly minimized. Here, we have considered single and exact pattern matching approach.

The rest of the paper is organized as follows: Section 2 presents the methodology and the implementation of methodology. The experimental environment and performance analysis is described in section 3. Section 4

discusses and concludes the paper.

2. METHODOLOGY

We propose a different approach to pattern matching, Double Hashing algorithm, which uses hashing technique. The main idea of our method is to compute hash value of each possible *m*-character substring in the text string that may match with the hash value of the pattern string. We can calculate the hash functions in linear time with help of some mathematical operations. Our target is to find each occurrence of the pattern string into the main DNA sequence, *i.e.*

Input pattern string:

G	C	T	A	A

Searching pattern into text:

A	C	C	G	C	T	A	A	T	C	G	C	T	A	A	C

Fig. 1. Example of single and exact pattern matching where two matches are found

Since, hashing is a time efficient technique based on data structure and it follows preprocessing, the search time of any pattern is possible in constant complexity. In this technique, hash function generates hash value which is less than the Mod value. However, if the number of generated hash values is large, then there is a high probability that two or more different substring obtaining the same hash value and so, searching a pattern may produce unexpected result. To reduce this problem at a considerable percentage, we have come up with a technique that generates hash values based on two different mod value for any particular substring. This technique is called double hashing and it can be efficiently used in DNA pattern matching.
The following subsections describe the constraint, dataset selection and implementation of the algorithms in details.

2.1. Constriaints Selection

We have considered specific constraints for both hashing and double hashing algorithm. Here,
DNA sequence T, $|T|$ = 1553927
Pattern P, $|P|$ = 10 $\leq |P| \leq 10^6$ ($|P|$ is multiple of 10)
Where $|S|$ denotes the length of any string S.

For hashing,
Base, B = 251
Mod, M = 10^9+7

For double hashing,
Two bases, B_1 = 251 & B_2 = 257
Two mod values, M_1 = 10^9+7 & M_2 = 10^9+9

2.2. Dataset Selection

In our experiment, we have used a real DNA sequence collected from NCBI database named NC_008229.1. This dataset contains the DNA sequence of the *Helicobacter acinonychis* with 1553927 base pairs (A,C,T and G) in total. The dataset is in FASTA format which includes the information of Genome nature such as accession number, reference number and species name. We ignored these particular information and considered only the full sequence of base pairs.

2.3. Algorithm Implementation

Algorithm 1 Hashing

Input: text string **T** and pattern string **P**
Output: report all the occuracnces of **P** in **T**

Preprocessing:
Create a linear table using base B and modular value M.
Array sequences will be as follows:
$Array[0] = B^0 \% M$
$Array[1] = B^1 \% M$
$Array[2] = B^2 \% M$
and so on..
Above sequences can be calculated more simply for any index i where $0 < i < n$
$Array[i] = (Array[i-1] * B) \% M$

Calculate an array Inv to store the modular inverse regarding the array values for division purpose.
According to combination of Euler's theorem [11] $A^{\phi(M)} \equiv 1 \ mod \ M$ and Fermat's little theorem [12] $A^{(M-1)} \equiv 1 \ mod \ M$.
Therefore, $\frac{1}{A} \% M \equiv 1 * A^{M-2} \% M$ and the Inv will be calculated as follows:
$Inv[i] = Array[i]^{M-2} \% M$
Calculate a linear hash table for the DNA sequence T as follows:
$Hash[0] = Array[0]$ and for any i when $0 < i < n$
$Hash[i] = (Hash[i-1] + T[i] * Array[i]) \% M$

Pattern Searching:
Let's assume x is the hash value from index i to index j for the DNA sequence T where $0 < i \leq j < n$.
So, $x = \{(Hash[j] - Hash[i-1]) * Inv[i]\} \% M$

for $i = 0 \rightarrow n - m$
  if $Hash\ (i \rightarrow i + m - 1) = Hash_p$
    report the occurance at index i
  end if
end for
end algorithm

Algorithm 2 Double Hashing

Input: text string **T** and pattern string **P**
Output: report all the occuracnces of **P** in **T**

Preprocessing:
Create two linear tables using base B_1, B_2 and modular value M_1, M_2 respectively.
Array sequences will be as follows:
$Array_1[0] = B_1^0 \% M_1$
$Array_1[1] = B_1^1 \% M_1$
$Array_1[2] = B_1^2 \% M_1$
and so on..
Similarly, calculate values for $Array_2$ using B_2 and M_2.
Above sequences can be calculated more simply for any index i where $0 < i < n$
$Array_1[i] = (Array_1[i-1] * B_1) \% M_1$
$Array_2[i] = (Array_2[i-1] * B_2) \% M_2$

Calculate two arrays Inv_1 and Inv_2 to store the modular inverse regarding the array values for division purpose.
According to combination of Euler's theorem $A^{\phi(M)} \equiv 1 mod M$ and Fermat's little theorem $A^{(M-1)} \equiv 1 \ mod \ M$. Therefore, $\frac{1}{A} \% M \equiv 1 * A^{M-2} \% M$ and Inv_1, Inv_2 will be calculated as follows:
$Inv_1[i] = Array_1[i]^{M_1-2} \% M_1$
$Inv_2[i] = Array_2[i]^{M_2-2} \% M_2$
Calculate two linear hash tables of the DNA sequence T as follows:
$Hash_1[0] = Array_1[0]$
$Hash_2[0] = Array_2[0]$
and for any i when $0 < i < n$
$Hash_1[i] = (Hash_1[i-1] + T[i] * Array_1[i]) \% M_1$
$Hash_2[i] = (Hash_2[i-1] + T[i] * Array_2[i]) \% M_2$

Pattern Searching:
Let's assume, x_1, x_2 are the hash values from index i to index j of the DNA sequence T where $0 < i \leq j < n$.
So, $x_1 = \{(Hash_1[j] - Hash_1[i-1]) * Inv_1[i]\} \% M_1$
$x_2 = \{(Hash_2[j] - Hash_2[i-1]) * Inv_2[i]\} \% M_2$

for $i = 0 \rightarrow n - m$
if $Hash_1 \ (i \rightarrow i + m - 1) = Hash_p$ & $Hash_2 \ (i \rightarrow i + m - 1) = Hash_p$
report the occurance at index i
end if
end for
end algorithm

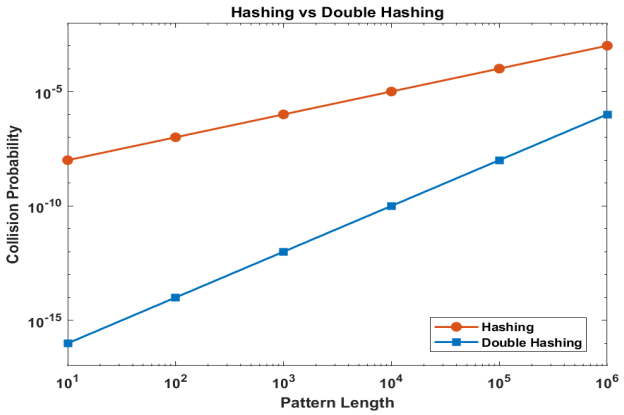

Fig. 2. Comparison of Hashing and Double Hashing with respect to collision probability

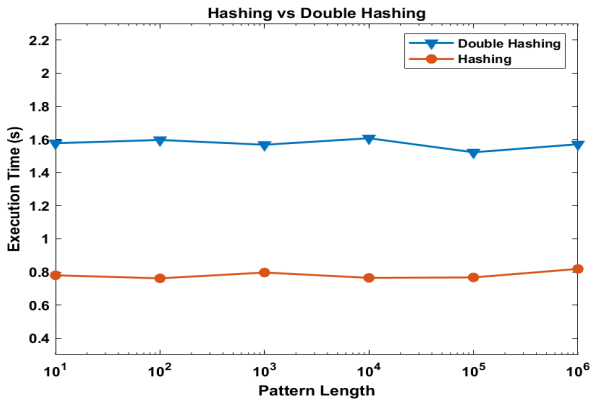

Fig. 3. Comparison of Hashing and Double Hashing with respect to execution time

3. EXPERIMENTAL RESULTS

The experimental result section is divided into two subsections. In subsection 3.1, the experimental setup is described. Subsection 3.2 gives a summary of the analysis of the performance.

3.1. Experimental Setup

The experiments on hashing and double hashing technique for pattern matching were run on 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz, 64-bit operating system, x64-based processor. Patterns were generated randomly from the dataset mentioned in subsection 2.2. In order to compare the algorithms more accurately, we have used files for storing and retrieving strings (text and pattern). The simulation was performed using C++ programming language. In each experiment, six patterns were searched from the main DNA sequence and the average of the findings were reported.

3.2. Performance Analysis

For analyzing the performance, we have used the pattern length of 10, 100, 1000, 10000, 100000, 1000000 and the main DNA sequence discussed in subsection 2.2. Two parameters are considered for comparison: probability of at least one collision and execution time. The probability of collision computed and presented in figure 2. we have used the concept of Birthday Paradox [13]. On basis of this paradox, probability of at least one collision in hashing is,

$$P_h = \frac{N-1}{M} \tag{1}$$

and in double hashing,

$$P_{dh} = \frac{(N-1)*(N-1)}{M_1 * M_2} \tag{2}$$

Here, M,M_1,M_2 are total possible hash values respectively and N is total generated hash value in the main DNA sequence. From our experiment, it is found that the probability of collision is less in double hashing than hashing technique for each pattern. To compare this probability, we use,

$$P_f = \frac{P_{dh} - P_h}{P_h}\% \tag{3}$$

where P_f is percentage increase in probability for double hashing, P_{dh} and P_h is the probability of collision for double hashing and hashing respectively. It is found that double hashing gives 99.89% less probability of collision on average than hashing.

Figure 3 shows the comparison of execution time between the algorithms. Here, the time is calculated in seconds. Both of the algorithms take linear execution time though, double hashing takes on average 0.79 seconds more time compared to hashing.

Due to some limitations in environment setup, there are some unpredictable values for each particular algorithm showed in both of the figures 2 and 3. To ensure the validity of the analysis, each experiment is repeated 10-12 times and the number of occurrences for each pattern in the main DNA sequence is also calculated and observed.

4. CONCLUSION

This paper introduced double hashing technique for achieving more accuracy by reducing the probability of collision compared to hashing technique. The experimental result shows that, to prevent hash values from colliding with one another, double hashing takes 0.79 second longer on average than hashing (both are linear time complexity) but it gives 99.89% less probability of collision compared to traditional hashing algorithm. Collision causes unexpected result while pattern matching that's why, less collision probability leads to higher accuracy and in many fields accuracy is important in DNA pattern matching. As, We have implemented our method of double hashing in single and exact pattern matching , our future research includes extending the double hashing technique for multiple as well as approximate pattern matching technique. We will also work to build a more efficient double hashing technique to reduce the execution time compared to hashing.

5. REFERENCES

[1] Raju Bhukya and DVLN Somayajulu, "Exact multiple pattern matching algorithm using dna sequence and pattern pair," *International Journal of Computer Applications*, vol. 17, no. 8, pp. 32–38, 2011.

[2] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein, *Introduction to algorithms*, MIT press, 2009.

[3] Ababneh Mohammad, Oqeili Saleh, and Rawan A Abdeen, "Occurrences algorithm for string searching based on brute-force algorithm," *Journal of Computer Science*, vol. 2, no. 1, pp. 82–85, 2006.

[4] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt, "Fast pattern matching in strings," *SIAM journal on computing*, vol. 6, no. 2, pp. 323–350, 1977.

[5] S Viswanadha Raju, KKVVS Reddy, and Chinta Someswara Rao, "Parallel string matching with linear array, butterfly and divide and conquer models," *Annals of Data Science*, vol. 5, no. 2, pp. 181–207, 2018.

[6] Robert S Boyer and J Strother Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no. 10, pp. 762–772, 1977.

[7] G Davies and S Bowsher, "Algorithms for pattern matching," *Software: Practice and Experience*, vol. 16, no. 6, pp. 575–601, 1986.

[8] Dan Gusfield, "Algorithms on stings, trees, and sequences: Computer science and computational biology," *Acm Sigact News*, vol. 28, no. 4, pp. 41–60, 1997.

[9] Jun-Ichi Aoe, Katsushi Morimoto, and Takashi Sato, "An efficient implementation of trie structures," *Software: Practice and Experience*, vol. 22, no. 9, pp. 695–721, 1992.

[10] Bruce J. McKenzie, Rodney Harries, and Timothy Bell, "Selecting a hashing algorithm," *Software: Practice and Experience*, vol. 20, no. 2, pp. 209–224, 1990.

[11] Yoshinori Fujisawa, Yasushi Fuwa, and Hidetaka Shimizu, "Euler's theorem and small fermat's theorem," *Formalized Mathematics*, vol. 7, no. 1, pp. 123–126, 1998.

[12] Boris Koichu and Rina Zazkis, "Decoding a proof of fermat's little theorem via script writing," *The Journal*

of Mathematical Behavior, vol. 32, no. 3, pp. 364–376, 2013.

[13] Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa, and Koji Toyota, "Birthday paradox for multi-collisions," in *International Conference on Information Security and Cryptology*. Springer, 2006, pp. 29–40.