International Journal of Computer Applications (0975 - 8887)
Volume 180 - No.26, March 2018

Parallel DNA Sequence Approximate Matching with Multi-Length Sequence Aware Approach

Hadeel Alazzam
University of Jordan
Department of Computer Science
Amman, Jordan

Ahmad Sharieh
University of Jordan
Department of Computer Science
Amman, Jordan

ABSTRACT

DNA sequence approximate matching is one of the main challenges in Bioinformatics. Despite the evolution of new technology, there is still a need for new algorithms that accommodate the huge amount of Bioinformatics data. In this paper, a parallel n-gram approach is proposed with a method that is taking in mind the variety of DNA sequence lengths for approximate matching. The proposed approach showed a satisfiability result in terms of time complexity compared to parallel dynamic programming method.

General Terms

Approximate Matching, Pattern Matching

Keywords

DNA Sequence, Longest Common Sequence, N-gram, Parallel

1. INTRODUCTION

Bioinformatics refers to solving biological problems using computer technology, with the evolution of technology, the amount of Bioinformatics data has been increased significantly. One of the main issues in Bioinformatics era is DNA pattern matching. DNA contains basic information that specified species, and the characteristics of a particular species [1].

Pattern matching is one of the most challenge in computer science field, i.e. Intrusion detection, image processing, and information retrieval. Pattern matching is considered to find all occurrences of pattern in the source. String matching is used to find the match between string s and target text t [2].

Since DNA sequencing is determined through four bases: Adenine, Guanine, Cytosine, and Thymine (A, G, C, and T respectively). The DNA sequencing matching can be solved as similarly as the string matching. There are several algorithms that solve string matching in literature starting from brute force solution that require time complexity of $O(m2^n)$, where m, and n is the length of the two strings [3]. String matching can be divided into two categories: exact matching and approximate matching [4]. Finding an exact match is not that complicated compared to approximate matching due to a lot of factors such as (existing of extra letter, string shifting, etc.).

Approximate matching is much useful than exact matching in many cases, for instance, finding spelling mistakes, or if two words have the same origins (i.e. Deleting, and deleted) [5].
Despite the technology evolves, the amount of data is increased significantly, and this increased the need to revise a new efficient algorithm to speed up the search of matching DNA sequences in a big dataset.

In this research, we propose a multi-threaded algorithm for solving a DNA sequence matching (Approximate matching) using n-gram. That aims to the reduce time complexity also in order to reduce the time needed to check a sequence in a pool of sequences in a dataset. The proposed method does a clustering of the dataset as a pre-processing step. This helps in reducing the number of comparing sequences significantly, and affects the overall time complexity.

The rest of this paper as follows; Section 2 presents the related works, while Section 3 discusses the methodology, Section 4 and Section 5 present the results and conclusion, respectively.

2. RELATED WORKS

Searching of exact or approximate pattern match is one of the fundamental problems in computer science. Many algorithms have been developed in the last three decades, focusing on the general problem [6][7] [8] [9] [10]. Also, the recent advances in genomics research, the speed of processor development growth, and the increased amount of data have been open the challenge in this era.
[6] Proposed a new hybrid algorithm for string matching (ASSBR) that compromises of Berry Ravindran (BR) shift function and Alpha Skip Search (ASS) algorithm to improve the performance. The hybrid algorithm tested by three types of data: DNA, Protein, and English text; and showed a better performance than both original algorithms in terms of number of attempts and character comparisons.
[7] Proposed a filter- then search- based exact matching algorithm for long pattern matching. The algorithm used the benefits of SIMD to accelerate the performance of the algorithm. The performance of the algorithm has a strong effective, especially when considering the length of the alphabet sizes. The algorithm has a good

performance for exact matching of long patterns in the worst case, the time complexity needed is O(n*m), for searching n byte pattern in m byte text.
[11] Proposed a two level n-gram approximate string matching that enhances the performance of n-gram exact string matching in [12] in term of query performance and reduce the index size by proposing an inverted index structure.
[13] Improved two matching algorithm EBOM/ FBOM, and proposed a two-new single pattern matching algorithm (SEBOM/ SFBOM), that removed unnecessary branches, and reduced the core calculation based on 2-grams algorithm. The algorithms are very fast for short pattern and reduced the number of memory accesses.
Bit parallel techniques have been used by many researchers in solving pattern matching [14] [15]. [14]developed a bit parallel algorithm based on a Shift-Or algorithm for exact and approximate string matching under hamming distance. The developed algorithm based on forward matching approach, which gives it?s a simplicity, is coming from the fact that pattern shifts are constant. The algorithm has a time complexity of worst case O(n).
At 1977, Boyer-More search algorithm have been proposed by [16], which is one of the string searching that is a standard benchmark for practical string search in literature. The algorithm based on preprocessing the pattern being searched for, not the text being searched for. The algorithm keeps in mind the information gathered during the preprocessing step, and uses it to skip sections in the text, thus reduces the time for searching comparing to other string search algorithms.
[15] Considered the bit parallel of Boyer-Moore type for an exact pattern match. They proposed four bit parallel algorithms. A two modification of Backward Nondeterministic DAWG Matching (BNDM) which is one of the proposed algorithm that achieved the best performance. It keeps the examined characters in history through a bit vector, while shifting based on this bit vector.
[9] Proposed a variation of the BNDM algorithm for exact matching, where at each alignment the algorithm performs a q-gram process, then tests the state variable. The algorithm variations were tested against several data: DNA, Binary, and English text with several lengths. The proposed variation of BNDM runs faster than other previous exact matching algorithms. Also, the new variation work fine with short patterns, which is not typical with Boyer Moore algorithm.
[10] Proposed a hybrid string matching algorithm SSTBMQS based on the best features of tunes Boyer-Moore algorithm and Quick- step algorithm. The algorithm was tested again many Benchmark datasets with different pattern length and different dataset size. It outperforms the original algorithm in terms of the number of comparisons and the number of attempts.
Despite the work in the related works, the amount of data is increased significantly. This increased the need to revise a new efficient algorithm to speed up the search. The proposed algorithm mixed between clustering for reducing the time complexity of searching, and using multi-threaded n-gram techniques, which make it a powerful tool for searching DNA sequences in large dataset.

3. METHODOLOGY

In this section, we describe the proposed approach for approximate matching DNA sequences. The methodology for our work divided into three parts: Clustering, Searching, and Compare Results.

Table 1. : n-gram example for different values of n.

Sample Sequence	Unigram	Bigram	Trigram
…AGCTTCGA…	…, A, G, C, T, T, C, G, A, …	…AG, GC, CT, TT, TC, CG, GA, …	…, AGC, GCT, CTT, TTC, TCG, CGA, …

3.1 Clustering the DNA Dataset (Preprocessing step)

The aim of clustering the dataset is to minimize the number of comparisons, where the target sequence will be searched only in the cluster of best match among other clusters. K-means clustering techniques used for clustering the dataset.
DNA sequences are converted to a binary code in order to represent it as vector, where $A = 00b, C = 01b, G = 10b, and T = 11b$. Every sequence is represented as a vector, and cosine similarity measure is used to find the similarity between the vectors. First, five sequences are chosen randomly and initially become the centroid of each cluster, then pick randomly a new sequence to find the best similarity between the sequence and each centroid. The sequence will be a part of the cluster of best match among all clusters. This procedure is repeated until the centroid values have no change. This procedure is repeated until all sequences are clustered. Algorithm 1 shows the pseudocode for K-means clustering, where Equation 1 presents the cosine similarity formula [17].

$$\begin{aligned} cosine\, similarity &= \frac{A.B}{||A||||B||} \\ &= \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \end{aligned} \quad (1)$$

Where A_i and B_i are components of vector A and vector B, respectively.

Algorithm 1 K-means Clustering Pseudocode (Sayed, 2010).

```
1: procedure K-MEANS CLUSTERING
2:     Clusters the data into k groups where k is predefined.
3:     Select k points at random as cluster centers.
4:     Assign objects to their closest cluster center according to
       the Cosine Similarity function.
5:     Calculate the centroid or mean of all objects in each cluster.
6:     Repeat steps 2, 3 and 4 until the centroids are not changing.
```

3.2 Approximate Matching

In order to enhance the speed of matching, the proposed approach uses a multithread n-gram to find the longest common subsequence between the target sequence and the sequences in the dataset. We use multiple values of n-gram $(2, 3, 4, 5, 6, 7, and\, 8)$ to decide the best value for n. Table 1 shows the n-gram for particular sequences with different values of n.

3.3 Compare results

The main challenge in approximate DNA matching in a dataset with various sequence's length, is that the alphabet of DNA sequence is only four characters $\{A, T, C, and\, G\}$; which means that the occurrence of these characters in particular order will occasionally happen, especially in the longest sequence in the dataset. So, here the longest common subsequence of a given DNA sequence will be always in the longest sequence in the dataset. Example 1 clarifies the point of approximate matching for a target DNA sequence with two DNA sequences that varies with their length.

Example 1.

Suppose that we have target DNA sequence, we have to check it with two DNA sequences that varies with their length. The length of the target sequence is 30 characters, the length of sequence 1 is 42 characters, while the length of sequence 2 is 493 characters. By finding the longest common subsequence(LCS) for target DNA with the other sequences, the LCS for sequence 1 equals to 15, while the LCS for sequence 2 equals to 30 which is the length of the target DNA. Initially, if we look to the red character at the sequences which represents the matching character, we can configure that sequence 1 has better matching than sequence 2; which is violating the LCS values. To solve this problem, the number of mismatch character in the match area has to be taken in mind. To do so, the matching area has to be cut, as shown in the example, then count the number of mismatching character along the matching area, and take the ratio of matching character to the mismatching character. This will give a reasonable result with respect to the length of DNA sequence.
Target DNA:
ATGAACCGCATCAGCACCACCACCATTACCACCAT
Sequence 1:
→*ATGTTTAAGACCGAGCAATCAAATTGC*←*TTTTAGAGACGG*
CCG
Sequence 2:
→*ATGCTTACGCCAAAACCCACGCTCGAATGTTGCACTGCGA*
TGCCGCCTATCGGGAAAACCCAACGGCGCTTTTCACCAGGT
TCGCGGCGATCGCCCGGCAACGCAGCCTGTGGAAT←*CCG*
CGGATATCGACAGTAAAGATGATTTAAAAAGCCTGCTGCTGG
TAGATAGCGCGCTGCGCATTACCGCTTTAGGTGACACTGTCA
CCATTCAGGCGTTATCTGATAATGGCGCCTCGTTATTGCCGC
TACTGGATACCGCCCTGCCCGCTGGCGTGGACGATGTCCTG
CCTGCCGGTCGCGTTCTACGCTTCCCGCCCGTCAGCCCATT
ATTAGATGAAGACGCCCGTTTATGCTCTCTGTCGGTATTTGAT
GCGTTCCGTCTGTTACAGGGAGTGGTGAACATACCGACGCA
AGAGCGGGAGGCTATGTTTTTCGGCGGTCTGTTTGCCTACG
ACCTGGTCGCTGGCTTTGAAGCGCTGCCACACC

$$Ratio\,of\,approximate\,match\,for\,sequence\,1 = \frac{match}{mismatch} = \frac{15}{12} = 1.25$$

$$Ratio\,of\,approximate\,match\,for\,sequence\,2 = \frac{match}{mismatch} = \frac{35}{86} = 0.406$$

According to the ratio values, sequence 1 has a better matching than sequence 2 with respect to DNA sequence lengths.

3.4 Algorithm Analysis

According to Algorithm 2, that presents the pseudocode of parallel n-gram, the time complexity of steps 1,2,4 is O(1), step 3 will execute n/p times, where n is the number of sequences in Dataset D, and p is the number of threads. Steps 5,6,7 will execute once, and step 8 will execute k times, where k is the number of n-gram in the target sequence. The overall time complexity for parallel n-gram is $O(\frac{n}{p} * k)$ for searching a sequence of length k, in a dataset contains n sequences.
The n is reduced according to the clustering preprocessing step. Then n is the number of sequence in a cluster only not the full dataset, where the input is the target sequence and a dataset to search through. The output will be the best matching sequence for the target sequence in the dataset.

Algorithm 2 Pseudocode for parallel n-gram

```
procedure PARALLEL N-GRAM
Input: Target Sequences TS, and Dataset D.
Outpu: Best match sequence from the dataset.
    1. string TS; // the Target sequence.
    2. let NTS = n-gram (TS, n); /* where n is the size of the n-gram */
    3. Spawn for each (S in D) /* where S is a sequence in dataset D */
    [match, dismatch]=Find-Match-Dismatch(S, NTS, n);
    4. end for
Find-Match-Dismatch(S,NTS,n){
    5. let NS= n-gram(S,n);
    6. let match =0;
    7. let mismatch=0;
    8. for each (g in NTS) // where g is an n-gram word
    9. if (NS.contains(g))
    10. ++match;
    11. NS.remove(g);
    12. Else if (match ¿ 0)
    13. ++mismatch
    14. end for
    15. return [match, dismatch]
    16. }
```

Table 2. : NCBI Datasets information [18].

S. No	Dataset	Species	Length in bp	Size Bytes
1	AC_000133.1	Homo sapiens chromosome 1	219,475,005	222,610,362
2	AL954800.2	Human chromosome 14	87,191,216	88,436,873
3	BA000007.2	Escherichia coli O157	5,498,450	5,577,087
4	AE006468.1	Salmonella enterica	4,857,432	4,926,935
5	NC_010473.1	E. coli?	4,686,137	4,753,183
6	NC_000913.3	E. coli?	4,641,652	4,706,047
7	NC_007146.2	Haemophilusinfluenzae	1,914,490	1,941,932
8	NC_008229.1	Helicobacter acinonychis	1,553,927	1,576,223
9	NC_001139.9	Saccharomyces cerevisiae	1,090,940	1,106,622

4. EXPERIMENTS AND RESULTS

This section shows the experimental results for the parallel n-gram. In order to compare our results, we implement parallel dynamic programming by [19] to find the LCS of the target sequence and compare the results to the parallel n-gram. The two algorithms are tested with and without clustering. A real DNA sequences used to test the algorithms, Table 2 presents the dataset used. The datasets obtained from the NCBI database [20].
All experiments were run on an Intel Core i5, with CPU 2.3 GHz, 10 GB of memory installed, running on MacOS High Sierra (version 10.13.1). The target sequence was generated randomly to test the approaches. The sequence length for testing were extracted from the sequence length in the datasets, where five lengths were picked; minimum sequence length through the datasets, the first quartile, the second quartile, the third quartile, and the maximum length among the datasets. Table 3 shows the sequence length used for testing.
Table 4 shows the running time in seconds for n-gram, and dynamic programming for three sequence lengths (short, medium,

Table 3. : The length of target sequences.

	Min sequence length	**First Quartile (Q1)**	**Second Quartile (Q2)**	**Third Quartile (Q3)**	**Max sequence length**
# of Characters	45	462	801	1245	16680

Table 4. : Running time in seconds for varies sequences length over nine NCBI datasets.

Dataset	Approach	Run Time/Seconds for each Sequence length				
		Min	Q1	Q2	Q3	Max
AC_000133.1	N-gram	0.871	1.672	0.866	2.286	3.518
	dynamic	1.014	10.198	6.392	22.621	140.821
	Cluster-N-gram	0.203	0.732	0.705	0.741	1.750
	Cluster-dynamic	0.253	4.286	3.805	5.934	55.083
AL954800.2	N-gram	0.088	0.144	0.160	0.201	0.373
	dynamic	0.099	0.446	0.430	1.102	5.779
	Cluster-N-gram	0.068	0.131	0.082	0.136	0.249
	Cluster-dynamic	0.081	0.337	0.283	0.602	4.647
BA000007.2	N-gram	1.021	1.921	0.893	2.374	4.891
	dynamic	1.168	13.447	7.290	24.242	162.745
	Cluster-N-gram	0.289	0.542	0.790	1.490	1.511
	Cluster-dynamic	0.326	1.891	4.037	9.510	38.087
AE006468.1	N-gram	0.261	0.476	0.313	0.681	0.789
	dynamic	0.280	3.138	1.878	6.807	33.374
	Cluster-N-gram	0.136	0.298	0.068	0.178	0.481
	Cluster-dynamic	0.150	2.111	0.257	0.741	4.855
NC_010473.1	N-gram	1.654	2.729	1.073	4.440	5.727
	dynamic	1.769	17.507	9.963	51.160	271.438
	Cluster-N-gram	0.348	1.536	0.488	2.711	2.429
	Cluster-dynamic	0.409	7.537	2.175	17.327	67.463
NC_000913.3	N-gram	0.366	0.942	0.546	1.611	1.180
	dynamic	0.525	5.439	3.199	12.958	76.671
	Cluster-N-gram	0.180	0.654	0.328	0.533	0.512
	Cluster-dynamic	0.204	3.240	1.714	5.375	13.898
NC_007146.2	N-gram	0.687	1.674	2.526	2.610	3.074
	dynamic	0.730	8.745	5.742	19.681	105.042
	Cluster-N-gram	0.235	0.815	0.506	1.484	1.154
	Cluster-dynamic	0.411	4.178	2.269	9.797	29.722
NC_008229.1	N-gram	0.596	1.626	0.753	2.368	3.062
	dynamic	0.645	7.826	6.459	16.838	98.093
	Cluster-N-gram	0.160	1.023	0.189	1.116	0.881
	Cluster-dynamic	0.178	5.323	0.755	9.397	15.711
NC_001139.9	N-gram	1.181	3.319	1.249	4.049	5.705
	dynamic	1.349	17.551	11.021	60.154	261.250
	Cluster-N-gram	0.361	1.013	0.614	1.599	2.128
	Cluster-dynamic	0.383	4.949	3.803	10.230	53.490

and long). To assure fair comparison, a random sequence generator was used to generate a random target sequence. Both algorithms run in multithreaded mode, and run in two scenarios: one without clustering, and other with clustering (preprocessing step). As the results show that the parallel n-gram, and dynamic programming have a close performance when the sequence length was short as Min value (45 long), while the n-gram have a significant enhancement in time, when the sequence length becomes larger. The performance of n-gram in term of time has been tested on nine different datasets from NCBI [20].
To evaluate the performance of using clustering as a preprocessing step, 10 different sequences were tested for every sequence length, and see how many times the result without clustering were the same as with clustering step. The result shows that 85
In order to evaluate the performance of the proposed approach in multithreading mode, a big dataset needed to seek the effectiveness for the number of threads on the proposed approach. To do so, all datasets from BCNI have been merged into one large dataset for evaluation purpose. Four versions of N-gram have been evaluated in multithreading mode, Figure 1 - Figure 5 shows the running time of N- gram version across one of eight numbers of threads for Min,

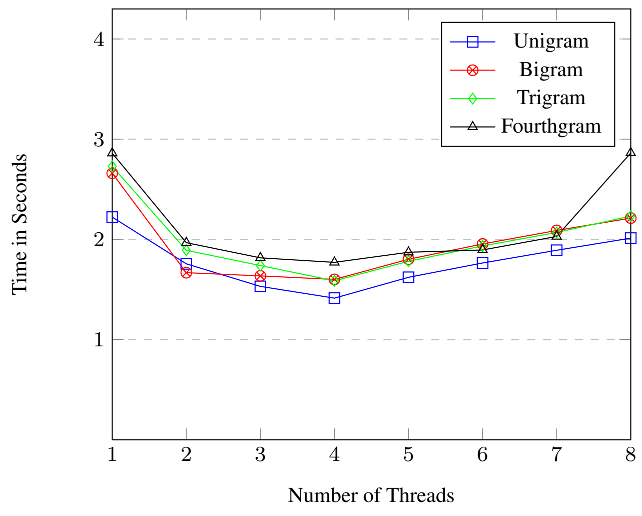

Fig. 1: Run time/second for n-gram versions with different number of threads for Min sequence length.

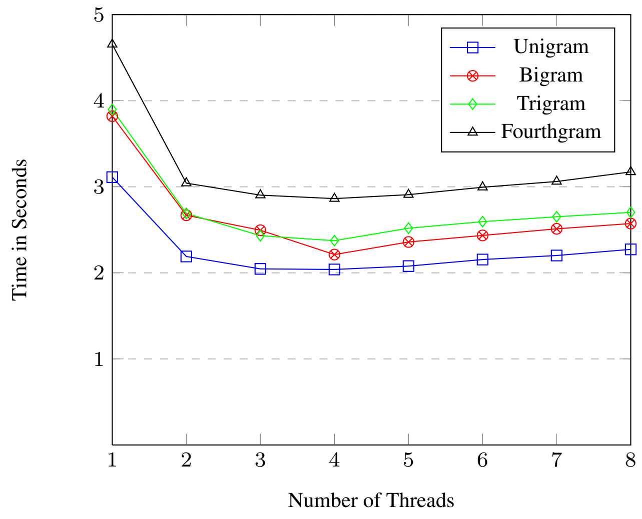

Fig. 2: Run time/second for n-gram versions with different number of threads for Q1 sequence length.

Q1, Q2, Q3, and Max target sequences length. The results show that using 4 threads is the optimal way of using multithreading. It's clear that the running time of using three threads is close to the running time of using four threads, this due to, that the fourth thread has been used for the main class (distributing the sequences among threads, and compare the results of the matching ratio among all sequences), as the results show that using more than four threads increased the running time, this due to the number of cores used and context-switching time.

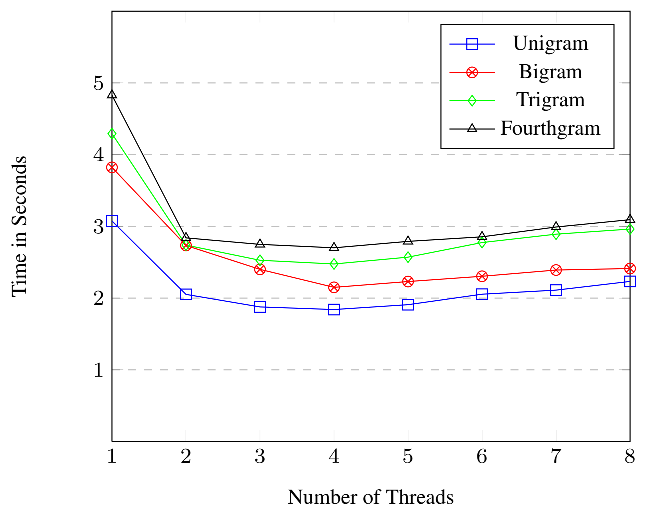

Fig. 3: Run time/second for n-gram versions with different number of threads for Q2 sequence length.

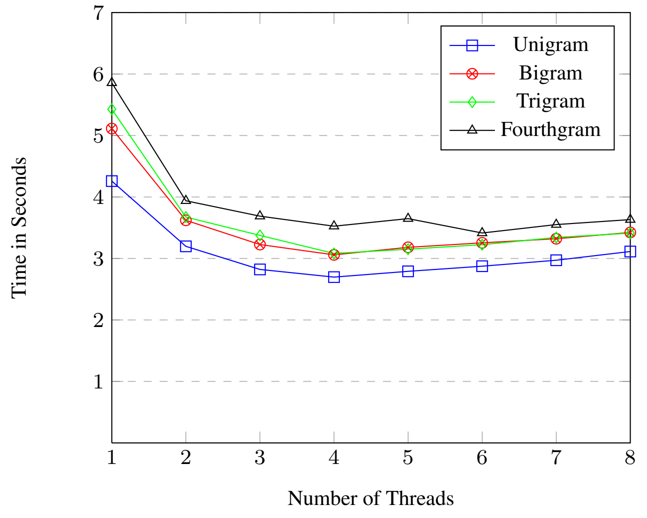

Fig. 4: Run time/second for n-gram versions with different number of threads for Q3 sequence length.

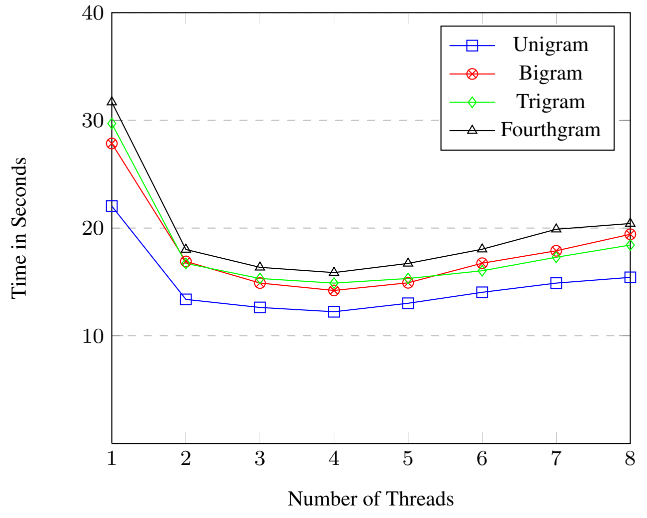

Fig. 5: Run time/second for n-gram versions with different number of threads for Max sequence length.

5. CONCLUSION AND FUTURE WORKS

One of the main issues of matching methods is the variation of sequences length in a particular dataset, that will affect the results. Since the longest sequence among the other will always have the longest common subsequence. In this paper, a parallel n-gram method has been proposed with multi-length sequences aware approach. The proposed method, used clustering method as preprocessing step to minimize the number of sequences comparative, however it absolutely affects the matching accuracy, since only one of the clustered will be checked. Parallel n-gram have been compared to parallel dynamic programming, and the results showed the parallel n-gram reduced the time for matching significantly when the sequence length become longer, whereas both algorithms have similar performance in terms of time complexity when the sequence length were short.
For future work, this method can be applied to string matching in general, also the proposed algorithm uses the prefix method, and can be enhanced by using both prefix and suffix methods together.

6. REFERENCES

[1] M.I. Khalil. Locating all common subsequences in two dna sequences. *Information Technology and Computer Science*, 5:81–87, 2016.

[2] Diao Y. Gyllstrom-D. Agrawal, J. and N. Immerman. Efficient pattern matching over event streams. . *In Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 147–160, June 2008.

[3] R. Bhukya and D. V. L. N. Somayajulu. Exact multiple pattern matching algorithm using dna sequence and pattern pair. *International Journal of Computer Applications*, 17(8):32–38, 2011.

[4] N. Singla and D. Garg. String matching algorithms and their applicability in various applications. *International journal of soft computing and engineering*, 1(6):218–222, 2012.

[5] J. Kawulok. Approximate string matching for searching dna sequences. *International Journal of Bioscience, Biochemistry and Bioinformatics*, 3(2):145, 2013.

[6] A. A. Almazroi. A fast hybrid algorithm approach for the exact string matching problem via berry ravindran and alpha skip search algorithms. *Journal of Computer Science*, 7(5):466, 2011.

[7] M. O. Kulekci. Filter based fast matching of long patterns by using simd instructions. *In Stringology*, pages 118–128, August 2009.

[8] Mustafa I. S. Sharieh, A. A. A. and N Obeid. Row column diagonal using multithreads for sequence alignment in dna. *European Journal of Scientific Research*, 30(1):6–25, 2009.

[9] Holub J. Peltola H. Durian, B. and J. Tarhio. Improving practical exact string matching. *Information Processing Letters*, 110(4):148–152, 2010.

[10] Naser M. A. S. Al-Dabbagh, S. S. M. and N. H. Barnouti. Fast hybrid string matching algorithm based on the quick-skip and tuned boyer-moore algorithms. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, 8(6):117–127, 2017.

[11] Whang K. Kim, M. and J Lee. n-gram/2l-approximation: a two-level n-gram inverted index structure for approximate string matching. *Computer Systems Science and Engineering*, 22(6):365, 2007.

[12] Whang K. Y. Lee J. G. Kim, M. S. and M. J Lee. n-gram/2l: A space and time efficient two-level n-gram inverted index structure. *In Proceedings of the 31st international conference on Very large data bases*, pages 325–336, August 2005.

[13] Yao N. Fan, H. and H. Ma. Fast variants of the backward-oracle-marching algorithm. *Fourth International Conference on*, 34:56–59, December 2009.

[14] K. Fredriksson and S. Grabowski. Practical and optimal string matching. *In SPIRE*, 3772:376–387, November 2005.

[15] H. Peltola and J. Tarhio. Alternative algorithms for bit-parallel string matching. *In SPIRE*, 2857:80–94, January 2003.

[16] R. S. Boyer and J. S Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[17] Gelbukh A. Gmez-Adorno H. Sidorov, G. and D. Pinto. Soft similarity and soft cosine measure: Similarity of features in vector space model. *Computacin y Sistemas*, 18(3):491–504, 2014.

[18] Sardaraz M. Tahir, M. and A. A. Ikram. Epma: Efficient pattern matching algorithm for dna sequences. *Expert Systems with Applications*, 80:162–170, 2017.

[19] M. V. Ramakrishnan and M. S. Eswaran. Acomparative study of various parallel longest common subsequence (lcs) algorithms. *International Journal of Computer Trends and Technology*, 4(2), 2013.

[20] R. C NCBI. Database resources of the national center for biotechnology information. *FNucleic acids research*, 45(D1):56–59, 2017.