Novel Algorithms for Finding the Closest l-mers in Biological Data

Xingyu Cai*
Dept. of CSE, Univ. of Connecticut

Abdullah-Al Mamun*
Dept. of CSE, Univ. of Connecticut

Sanguthevar Rajasekaran †
Dept. of CSE, Univ. of Connecticut

***Abstract*—With the advances in the next generation sequencing technology, huge amounts of data have been and get generated in biology. A bottleneck in dealing with such datasets lies in developing effective algorithms for extracting useful information from them. Algorithms for finding patterns in biological data pave the way for extracting crucial information from voluminous datasets. In this paper we focus on a fundamental pattern, namely, the closest l-mers. Given a set of m biological strings $S_1, S_2, \ldots, S_m$ and an integer l, the problem of interest is that of finding an l-mer from each string such that the distance among them is the least. I.e., we want to find m l-mers $X_1, X_2, \ldots, X_m$ such that X_i is an l-mer in S_i (for $1 \leq i \leq m$) and the Hamming distance among these m l-mers is the least (from among all such possible l-mers). This problem has many applications. An application of great importance is motif search. Algorithms for finding the closest l-mers have been used in solving the (l, d)-motif search problem (see e.g., [1], [2]). In this paper novel exact and approximate algorithms are proposed for this problem for the special case of $m = 3$. We consider the Euclidean distance metric if the sequences contain real numbers.**

***Index Terms*—Closest l-mers; Closest triplet; Efficient algorithms; Randomized algorithms; Time series motifs; (l, d)-motifs**

I. Introduction

Large amounts of data get generated in every area of science and engineering. This is especially true in the biological domain. Currently, the bottleneck is not in generating data but is in processing these data. Efficient big data analytics algorithms are called for. A powerful analytics paradigm is patterns finding. In this paper we study an important pattern that can be used to solve many other problems including motif search. Specifically, we investigate the problem of finding the closest l-mers in an input of strings. The biological strings could be DNA sequences, protein sequences, etc. Algorithms for finding the closest l-mers have been used to solve the (l, d)-motif search problem, see for example [1], [2].

The pattern finding problem of interest can be stated as follows. The input are m biological sequences $S_1, S_2, \ldots, S_m$, each of length n, and an integer l. The problem is to find m l-mers $X_1, X_2, \ldots, X_m$ such that X_i is in S_i (for $1 \leq i \leq m$) and the Hamming distance among these l-mers is the least (from out of all such l-mers). X is an l-mer in a sequence S if X is a subsequence of S of length l. Each input sequence can be thought of as a string of characters from a finite alphabet Σ. For instance, each input sequence could be a DNA sequence or a protein sequence. We refer to this pattern finding problem as the closest l-mers problem (CLP). If $X_i = x_1^i x_2^i x_3^i \ldots x_l^i$, for $1 \leq i \leq m$, are any l-mers, then the Hamming distance among them is defined as $\sum_{j=1}^{l} d(x_j^1, x_j^2, \ldots, x_j^m)$ where $d(x_j^1, x_j^2, \ldots x_j^m)$ is zero if all of the characters $x_j^1, x_j^2, \ldots$, and x_j^m are the same; and $d(x_j^1, x_j^2, \ldots x_j^m)$ is 1 otherwise. If the input consists of time series data, many possibilities arise. Consider the case of $m = 3$. Let X, Y, and Z be any three l-mers. Then, one possible distance among these three is the **pairwise-sum distance** $d(X, Y, Z) = d(X, Y) + d(Y, Z) + d(Z, X)$ where $d(X, Y)$ is the Euclidean distance between X and Y.

A special case of the CLP when $m = 2$ has been studied in the literature before. For instance, [1] show that this problem can be solved in $O(n^2)$ time for $m = 2$, where n is the length of each of the two input sequences. Note that a trivial algorithm to solve this problem will examine each pair of l-mers A and B where A comes from the first sequence and B comes from the second sequence, compute the Hamming distance between A and B, and output the pair of l-mers with the least distance. This brute force algorithm runs in time $O(n^2 l)$. The $O(n^2)$-time algorithm has been used in solving the (l, d)-motif search problem (see e.g., [1], [2]). Time series motif mining could be viewed as a special case of CLP, and many algorithms have been recently used to solve this problem, such as FFT technique in [3] and $O(n^2)$ methods in [4] [5], and embedding-based approach in [6].

The case of $m > 2$ is very important as well. For instance, in the case of (l, d)-motif search, an algorithm for the case of $m > 2$ can be used in the algorithms of [1], [2] in which case the performance of these algorithms will improve. Also, for the time series motif mining problem, m being more than 2 can provide deeper insights.

In this paper we present novel algorithms for solving the CLP when $m = 3$. We refer to this special case of the CLP as the **closest triplet** problem. Specifically, we offer three different algorithms. Two of these are exact and the third one is approximate. An algorithm is exact if it always outputs the closest l-mers. On the other hand, an approximate algorithm may not output the closest l-mers all the time. In general it outputs l-mers whose distance is very nearly the same as that of the closest l-mers. A closely related problem is one where the l-mers could come from the same sequence, and we also extend our algorithms to address this problem, by enforcing

*Co-first authors with equal contributions to this work

†Corresponding author. This work has been supported in part by the NSF grant 1447711.

one additional constraint that the l-mers should not overlap.

Paper Organization: The rest of this paper is organized as follows. In Section II we review an existing algorithm for CLP when $m = 2$. This special case is called the *closest pair of subsequences problem*. Next in Section III, we propose two exact algorithms. The first algorithm uses $O(n^2)$ multiplications and $O(n^3)$ addition operations, and uses $O(n^2)$ memory. We call this algorithm **Exact-0**. The second algorithm has a run time of $O(n^3)$, but only uses $O(1)$ memory. We call the second algorithm **Exact-1**. Another version of the second algorithm takes $O(n)$ memory but reduces the running time to $O(n^3 - n^2 l)$. Note that the second version only applies to pairwise-sum distances. In Section IV we present our experimental results. We have used both biological and time series data, and employed Hamming distance and pairwise-sum Euclidean distance, respectively. Section V provides some concluding remarks.

II. Background

The $O(n^2)$ Time Algorithm of [1]

For solving the closest pair of l-mers problem, Pevzner and Sze exploit the overlaps during the process of computing pairwise distances [1]. This eliminates the dependence of the run time on l [1]. Let $S = s_1, s_2, \ldots, s_n$ be any given sequence data and let l be the length of the subsequences we are interested in. The problem of finding the closest pair of subsequences in S can be decomposed to $(n - l + 1)$ subproblems. Let these subproblems be referred to as $\mathcal{P}_i$, for $1 \leq i \leq (n - l + 1)$. Each $\mathcal{P}_i$ computes the distance between the following pairs of subsequences of length l: $([s_j, s_{j+1}, \ldots, s_{j+l-1}], [s_{i+j-1}, s_{i+j}, \ldots, s_{i+j+l-2}])$, for $1 \leq j \leq (n - l + 1)$. Note that in these distance calculations, we can ignore any pair if the elements $s_{n'}$ (for $n' > n$) appear in any of the two subsequences. Let the distance between the pair $((s_j, s_{j+1}, \ldots, s_{j+l-1}), (s_{i+j-1}, s_{i+j}, \ldots, s_{i+j+l-2}))$ be d^i_j, for $1 \leq j \leq (n - l + 1)$.

[1]'s algorithm makes use of the overlaps in consecutive pairs. We use the Euclidean distance metric as an example here but it is easy to extend our discussion to Hamming distance as well. Since $(d^i_j)^2 = (s_j - s_{i+j})^2 + \ldots + (s_{j+l-1} - s_{i+j+l-1})^2$, the next pair's squared distance could be expressed as $(d^i_{j+1})^2 = (s_{j+1} - s_{i+j+1})^2 + \ldots + (s_{j+l} - s_{i+j+l})^2 = (d^i_j)^2 - (s_j - s_{i+j})^2 + (s_{j+l} - s_{i+j+l})^2$.

Clearly, the computation of $(d^i_1)^2$ takes $O(l)$ time. Note that $(d^i_j)^2$ can be obtained from $(d^i_{j-1})^2$ in an additional $O(1)$ time (for $j > 1$). Thus the problem $\mathcal{P}_i$ can be solved sequentially in a total of $O(n)$ time (for any specific value of i, $1 \leq i \leq (n-l+1)$). Since there are a total of n subproblems, the total running time is $O(n^2)$, which is independent of l.

III. Proposed Algorithms

When $m = 3$ we can solve the CLP in $O(n^3 l)$ time in a straight forward way. The idea is to compute the distance among every triplet of l-mers. For each triplet the time spent is $O(l)$ and there are $O(n^3)$ triplets.

A. Exact-0 Algorithm for Pairwise-sum Distances

We can solve the CLP for $m = 3$ in $O(n^3)$ time using the algorithm of [1] as a subroutine. This algorithm will work as follows: 1) Use the algorithm of [1] to compute pairwise distances in $O(n^2)$ time. Store all of these distances. Followed by this, compute the distance for each possible triplet of l-mers.

Theorem 3.1: We can use Exact-0 algorithm to solve the CLP for $m = 3$ using $O(n^2)$ multiplications and $O(n^3)$ addition operations, as well as $O(n^2)$ space. □

B. Exact-1 Algorithm

If the input size n is large, the $O(n^2)$ memory cost may be prohibitive. For example, when $n = 40 \times 10^3$, using double precision storage, the algorithm would require roughly 10 GB of memory. This is quite large. Besides, as memory usage increases, the memory accessing cost will become dominant and make the algorithm take longer time to finish.

Motivated by this, we have developed a memory efficient algorithm that solves this problem in $O(n^3)$ time with only a constant memory requirement. In the case of pairwise-sum distance measurement, $O(n^2 l)$ time could be saved at the cost of $O(n)$ memory. We thus have two versions: The first version takes $O(n^3)$ time and uses $O(1)$ memory; the second version takes $O(n^3 - n^2 l)$ time and employs $O(n)$ memory. The second version is very useful when l is not far less than n. For instance, if $l = 0.3n$, then 30% of the total running time could be reduced.

*1) **Version 1:** $O(1)$ **Memory**:* The key idea to reduce the memory cost from $O(n^2)$ to $O(1)$, is by exploiting the overlaps like in [1]. Rather than using [1]'s algorithm as a subroutine to compute all pairwise distances in the first step, we split the entire procedure into subproblems $\mathcal{P}_{ik}$ such that each subproblem represents a unique alignment (i, k) and outputs distances of the triplets $(a, a+i, a+i+k), a \in [1, n]$. Clearly, consecutively outputting the distance as a shifts, would cost $O(n)$ time for each subproblem, and there are a total of $O(n^2)$ subproblems. So the total running time for this algorithm is $O(n^3)$. Besides, since only one set of distances (for pairwise-sum distance, three pairwise distances are stored; for direct distance, one triplet distance is stored) needs to be stored in memory, the memory cost becomes $O(1)$ during the entire process. This can be seen as an enhanced version of [1]'s algorithm. We arrive at the following Theorem:

Theorem 3.2: The CLP can be solved in $O(n^3)$ time using $O(1)$ space applying Exact-1 algorithm version 1. □

*2) **Version 2:** $O(n)$ **Memory**:* Without loss of generality, we give an illustration using the example of finding the closest 3 l-mers from one single sequence under pairwise-sum measurement metric, with a constraint that there are no overlaps for l-mers in the closest triplet. In the previous $O(1)$ version, for each alignment $< i, k >$, the starting cost to compute $d(0, i), d(i, i+k), d(0, i+k)$ still requires $O(l)$ time each. And since there are $O(n^2)$ alignments, the subproblems' starting costs accumulate to $O(n^2 l)$. After starting, all the remaining distances are calculated in only $O(1)$ time. As a

result, removing the starting cost could save a descent fraction of the total running time. As noticed, the majority of starting cost is in the form of $d(0,i), i \in [l, n-2l]$. Thus a simple solution is to store these values in memory to avoid repetition in computing them. This only requires $O(n)$ storage and the running time is reduced to $O(n^3 - n^2 l)$ as a consequence. Details of this algorithm are given in Algorithm 1.

Algorithm 1 Exact-1 Algorithm with $O(n)$ Memory

Input: Sequence $A = s_1, s_2, \ldots, s_n$; subsequence A_t is defined as $A_t = [s_t, s_{t+1}, \ldots, s_{t+l-1}]$; $\hat{d}(A_{t_1}, A_{t_2})$ denotes **squared** Euclidean distance between A_{t_1}, A_{t_2}
Output: A triplet of subsequences that has the least pairwise-sum Euclidean distance

Set best-so-far $b = \infty$
for $i = 0$ to $n - l$ **do**
  Compute and store $D_1[i] = \hat{d}(A_0, A_i)$
end for
for $k = l$ to $n - l$ **do**
  Obtain $\hat{d}_1 \leftarrow D_1[k]$
  for $j = l$ to $n - k$ **do**
    Compute $\hat{d}_2 = \hat{d}(A_k, A_{k+j})$; Obtain $\hat{d}_3 \leftarrow D_1[k+j]$
    $tmp = \sqrt{\hat{d}_1} + \sqrt{\hat{d}_2} + \sqrt{\hat{d}_3}$
    if $tmp < b$ **then**
      update $b \leftarrow tmp$ and the corresponding indices
    end if
    for $i = 0$ to $n - l - k$ **do**
      $\hat{d}_1 = \hat{d}_1 - (s_i - s_{i+k})^2 + (s_{i+l} - s_{i+k+l})^2$
      $\hat{d}_2 = \hat{d}_2 - (s_{i+k} - s_{i+k+j})^2 + (s_{i+k+l} - s_{i+k+j+l})^2$
      $\hat{d}_3 = \hat{d}_3 - (s_i - s_{i+k+j})^2 + (s_{i+l} - s_{i+k+j+l})^2$
      $tmp = \sqrt{\hat{d}_1} + \sqrt{\hat{d}_2} + \sqrt{\hat{d}_3}$
      if $tmp < b$ **then**
        update $b \leftarrow tmp$ and the corresponding indices
      end if
    end for
  end for
end for
return b and the corresponding indices

Theorem 3.3: We can solve the CLP in $O(n^3 - n^2 l)$ time using $O(n)$ memory applying Exact-1 algorithm version 2. □

C. Approximate Algorithm

We also provide an approximate algorithm addressing CLP problem when $m = 3$. Due to page limits, the details of the approximate algorithm is in the full version of this paper.

IV. Experimental Evaluation

In this section, we evaluate our proposed algorithms for run time and/or accuracy using two existing datasets. Each dataset is tested using one measurement metric (Hamming distance and pairwise-sum Euclidean distance). The test platform we are using is equipped with Intel Xeon CPU @ 2.67GHz.

A. Genome Dataset

We have performed intensive experiments on human genome data set [7]. We chose 21 chromosomes and grouped them into 7 files each having 3 chromosome sequences. We have run Exact-1 and brute-force algorithms in order to identify the closest l-mers among three sequences in each set, and there are 7 sets of genome sequences. We have used different values for n ranging from 4,000 to 60,000. The first n elements of the 7 sets of genome sequences are used to form the input sequences. Hamming distance is used as the distance metric. For a fixed n and l, we call such a combination a test group, and the running time is calculated as an average over the 7 sets of genome sequences for this group.

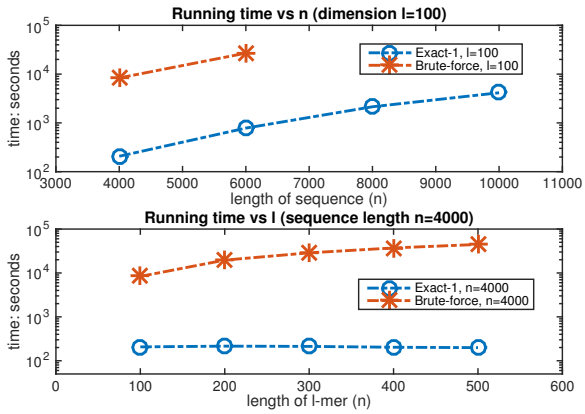

Fig. 1. Running time comparison on the Genome dataset

At first we compare our proposed algorithms with the ($O(n^3 l)$ time) brute-force algorithm. The result is shown in Figure 1 for n ranging from $4,000$ to $10,000$, with $l = 100$ to 500. The running time is provided in Semilog-Y plot for a better illustration. When n is larger than $6,000$, the brute-force algorithm takes more than 10 hours. Thus we have run the brute-force algorithm only for $n = 4,000$ and $n = 6,000$. From the plot, we clearly see that the Exact-1 algorithm outperforms the brute-force algorithm by more than one order of magnitude for all the 5 different l values. Also, as the dataset size n increases, the running time difference increases. For a fixed $n = 6,000$, increasing l will increase the running time of the brute-force algorithm by a lot, but almost has no effect on the Exact-1 algorithm. This is due to the fact that the running time of Exact-1 is independent of l as shown in a previous analysis.

B. Human Activity Dataset

In this experiment, we evaluate our algorithms under the pairwise-sum distance measurement using Euclidean distance, i.e., $d(A_i, A_j, A_k) = d(A_i, A_j) + d(A_j, A_k) + d(A_i, A_k)$. The goal is to identify 3 l-mers from one single sequence A, such that their pairwise-sum distance is minimum, under the constraint that they do not overlap with each other.

The dataset we use is from UCI Machine Learning Repository [8]. For a fair comparison, we have randomly selected one dataset which happens to be the Heterogeneity Activity Recognition Data Set [9]. This contains around 1×10^7 real numbers. This dataset includes cellphone accelerometer and gyroscope recorded data for human activity. There are 6 sensor coordinates in total and each forms a long sequence of numbers.

To perform evaluations, we downsampled the dataset with an interval of 10 for each sequence, and then applied a shifting of 0 and 5 to obtain a total of 12 downsampled sequences.

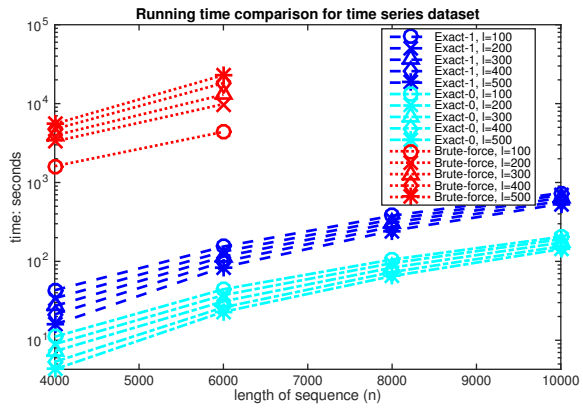

Fig. 2. Running time comparison on time series data (Activity dataset)

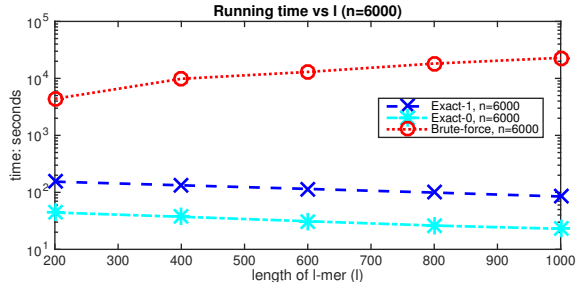

Fig. 3. Running time comparison for different dimensions (Activity dataset)

We have performed evaluations on different n and d values. The first n elements in each sequence have been pulled out to form a group of data sequences. The evaluation is based on the average performance across 12 data sequences in each group, and accordingly the accuracy is reported as the number of Hits out of 12. Three algorithms are evaluated on this dataset, which are Exact-0, Exact-1 and Brute-force.

Similar to our previous experiment, we evaluate the algorithms on different n and l values. As shown in Figure 2, the Brute-force algorithm is the most time consuming and it takes more than 10 hours in our experiment for $n > 6,000$ cases. Exact-0 gives the best run time. For datasets of size up to $10,000$, around 600MB memory is utilized by Exact-0. However, since Exact-0 is five times faster than Exact-1, it is very competitive on small to moderate datasets.

In the next test we demonstrate how the run times of the algorithms vary as l changes. We pick $n = 6,000$ and change l from 100 to 500. Figure 3 plots three curves representing the three algorithms, respectively. As expected, for both exact algorithms Exact-0 and Exact-1, the running time decreases as l increases, because the actual number of triplets $(n-l)^3$ decreases. For Brute-force, the runnig time increases due to its dependence on l.

The next experiment is performed on larger n and l values. In particular, $n = 10,000,\ 20,000$, d ranges from 200 to $2,000$. Brute-force algorithm is not included here as it takes too much time to finish. From Table I, we can see that Exact-0 is faster than Exact-1 as long as enough memory is given ($O(n^2)$ memory is required).

TABLE I
RUNNING TIMES ON LARGE DATASETS

		l=200	l=400	l=600	l=800	l=1,000
n=10k	Exact-0	167.5	139.4	108.2	88.4	69.0
	Exact-1	695.9	587.8	489.8	403.5	329.4
		l=400	l=800	l=1,200	l=1,600	l=2,000
n=20k	Exact-0	1,404.0	1,146.8	953.4	725.6	582.6
	Exact-1	5,583.9	4,742.6	3,997.6	3,247.1	2,681.1

C. Summary of Experimental Evaluation

In this section we have performed comprehensive evaluations on the Genome dataset and the Activity dataset. The measurement metrics we used are Hamming distance and pairwise-sum Euclidean distance. The experiments have been carried out for different n and l values. From the experiments we make the following observations: The performances are consistent using both measurement metrics on two different datasets, showing our proposed algorithms are robust; Exact-0 algorithm runs faster than Exact-1, at a cost of $O(n^2)$ memory. On small datasets, it is very competitive; Exact-1 is performing much better than brute-force, making it a good candidate for exact algorithm that always outputs the correct answer; Both exact algorithms' running times decrease as l increases. Due to page limits, experimental results on the approximate algorithm can be found in the full version of this paper.

V. Conclusions

In this paper we consider the problem of finding the closest l-mers. Our experimental results reveal that our algorithms are highly competitive.

References

[1] P. A. Pevzner, S.-H. Sze, *et al.*, "Combinatorial approaches to finding subtle signals in dna sequences.," in *ISMB*, vol. 8, pp. 269–278, 2000.

[2] J. Davila, S. Balla, and S. Rajasekaran, "Fast and practical algorithms for planted (l, d) motif search," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, vol. 4, no. 4, pp. 544–552, 2007.

[3] Y. Li, M. L. Yiu, Z. Gong, *et al.*, "Quick-motif: An efficient and scalable framework for exact motif discovery," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pp. 579–590, IEEE, 2015.

[4] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh, "Matrix profile i: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets," in *IEEE ICDM*, 2016.

[5] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh, "Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pp. 739–748, IEEE, 2016.

[6] P. Papapetrou, V. Athitsos, M. Potamias, G. Kollios, and D. Gunopulos, "Embedding-based subsequence matching in time-series databases," *ACM Transactions on Database Systems (TODS)*, vol. 36, no. 3, p. 17, 2011.

[7] K. D. Pruitt, G. R. Brown, S. M. Hiatt, F. Thibaud-Nissen, A. Astashyn, O. Ermolaeva, C. M. Farrell, J. Hart, M. J. Landrum, K. M. McGarvey, *et al.*, "Refseq: an update on mammalian reference sequences," *Nucleic acids research*, vol. 42, no. D1, pp. D756–D763, 2013.

[8] A. Asuncion and D. Newman, "Uci machine learning repository," 2007.

[9] A. Stisen, H. Blunck, S. Bhattacharya, T. S. Prentow, M. B. Kjærgaard, A. Dey, T. Sonne, and M. M. Jensen, "Smart devices are different: Assessing and mitigatingmobile sensing heterogeneities for activity recognition," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pp. 127–140, ACM, 2015.