On Multiple Longest Common Subsequence and Common Motifs with Gaps (Extended Abstract)

Suri Dipannita Sayeed(✉), M. Sohel Rahman, and Atif Rahman

Department of Computer Science and Engineering,
Bangladesh University of Engineering and Technology, Dhaka 1000, Bangladesh
suri.esha6@gmail.com, {msrahman,atif}@cse.buet.ac.bd

Abstract. Motif finding is the problem of identifying recurring patterns in sequences. It has been widely studied and several variants have been proposed. Here, we address the problem of finding common motifs with gaps that are present in all strings of a finite set. We prove that the problem is NP-hard by reducing the multiple longest common subsequence (MLCS) problem to it. We also provide a branch and bound algorithm for MLCS and show how the algorithm can be extended to give an algorithm for finding common motifs with gaps after common factors that occur in all the strings have been identified.

Keywords: Computational biology · Motif finding · Complexity
Branch and bound

1 Introduction

Motifs are recurring patterns in sequences and motif finding is a widely studied problem in computational biology. It has diverse applications for example in identifying co-expressed genes. Expression of a gene usually requires binding of a transcription factor in the promoter region. Presence of near identical sequences in promoter regions of genes indicates that they are regulated by the same transcription factor and are likely to be co-expressed.

In many cases, the motifs may not be identical and motif finding algorithms need to be robust to differences in sequences. In addition, motifs may not be contiguous. Considering these, the problem of finding common motifs with gaps was introduced [2,5]. Antoniou et al. gave an algorithm polynomial in length of strings and exponential in number of strings using finite automata and conjectured that asymptotically more efficient algorithms may not be possible [2]. Other variants of the problem with constraints on lengths of gaps have also been proposed and algorithms have been provided [1].

In this paper, we prove that the problem of finding common motif with gaps is NP-hard, for alphabet size of four or more, by reducing to it the multiple longest common subsequence (MLCS) problem, i.e., the problem of finding the longest

M. S. Rahman et al. (Eds.): WALCOM 2018, LNCS 10755, pp. 207–215, 2018.
https://doi.org/10.1007/978-3-319-75172-6_18

common subsequence among a set of sequences. MLCS is also a well studied problem in theoretical computer science and has applications in computational genomics. MLCS was proved NP-hard by Maier [7] and dynamic programming algorithms are known that run in time $O(n^d)$, for d sequences with maximum length, n [6]. Wang et al. [9] gave a dominant point based algorithm with the divide and conquer approach to compute the dominant points and designed a Quick-DP algorithm using those points and later Yang et al. [10] presented a progressive algorithm with efficient parallelization. Huang and Lim gave a branch and bound algorithm using minimum of pairwise longest common subsequence lengths as a bounding condition [4]. As MLCS problem is widely used in protein and genome sequence analysis, further improvement of the algorithm can contribute significantly in the studies of computational genomics [3].

In this paper, first we formally settle the question of complexity of the problem of finding common motifs with gaps. We also present a branch and bound algorithm for computing the longest common subsequences of a set of strings. We pre-process the sequences to explore more promising paths first and to speed-up the pruning process and we explore bounding strategies previously not considered [4]. We then extend this algorithm to find common motifs with gaps. We first find the common factors appearing in all the strings and then use a branch and bound algorithm to chain together the factors. Although this problem is treated as a hard problem in the literature and is handled accordingly, to the best of our knowledge this is the first attempt to prove the hardness thereof formally. And the reduction from MLCS allows us to adapt the branch and bound algorithm for MLCS to solve the problem of finding common motifs with gaps.

The rest of the paper is organized as follows. We give the problem definitions in Sect. 2 and prove the hardness of the problem of finding common motifs with gaps in Sect. 3. The algorithms are presented in Sect. 4 along with examples. Finally, in Sect. 5 we conclude the paper.

2 Background

A motif is a common pattern that appears frequently among a set of genome sequences. In this section we formally define the two problems that we are going to address in the rest of the paper.

2.1 Common Motifs with Gaps

Consider a set of strings $S = \{S_1, S_2, .., S_d\}$ over the alphabet $\Sigma = \{A, C, G, T\}$ and two integers p,q, where $1 \leq p \leq q \leq min(|S_j| : j \in \{1, \ldots, d\})$ are given. The problem of finding common motifs with gaps (CMG) aims at finding common words $P_1, P_2, ..., P_m$ such that: $P_1 *^{d_{i,1}} P_2 *^{d_{i,2}} \cdots *^{d_{i,m-1}} P_m$ occurs in S_i, for all $i \in \{1, \ldots, d\}$, $m > 1$, $p \leq |P_j| \leq q$ for all $j \in \{1, \ldots, m\}$ and $d_{i,j} \geq 1$ for all $i \in \{1, \ldots, d\}, j \in \{1, \ldots, m-1\}$. Here, $*$ is the don't care symbol that matches any character in Σ.

For example, given a set of three strings $S = S_1, S_2, S_3$ and minimum factor size, $p = 1$ and maximum factor size, $q = 2$, the substrings AC, AA and CA form the common motifs that satisfy the required criteria as highlighted below:

$$S_1 = \mathbf{AC}A\mathbf{AA}A\mathbf{CA}CAAA$$
$$S_2 = AC\mathbf{AC}C\mathbf{AA}C\mathbf{CA}CA$$
$$S_3 = C\mathbf{AC}A\mathbf{AA}C\mathbf{CA}CCA$$

In the optimization version of the problem, we want to maximize m i.e. we seek a common motif with gaps with maximum number of factors and in the decision version of the problem, for a given m, we want to check whether there is a common motif with gaps with $\geq m$ factors.

2.2 Multiple Longest Common Subsequence

The Multiple Longest Common Subsequence (MLCS) problem aims at finding a longest subsequence shared among a set of sequences. Let, $S = \{S_1, S_2, S_3,...,S_d\}$ be a set of sequences over a finite alphabet Σ. The Longest Common Subsequence (LCS) of set S is a sequence s with length ℓ, such that it is of the highest length among all subsequences that are shared among all S_i, $i \in \{1, \ldots, d\}$. For example,

$$S_1 = informatics$$
$$S_2 = bioinformatics$$
$$S_3 = proteomics$$

One of the subsequences for this example is $s_1 = mics$, one is $s_2 = tics$ and another is $s_3 = omics$. The longest one is $s_3 = omics$. Notably, this sequence may not necessarily be unique.

In the optimization and decision versions of the problem we intend to find an LCS of the maximum length and decide if there is an LCS greater or equal to a given length, respectively.

3 Complexity of Common Motifs with Gaps

Theorem 1. *CMG problem is NP-complete.*

Proof. To show that the decision version of $CMG \in NP$, for a given set $S = \{S_1, S_2, \ldots, S_d\}$ of sequences, a sub-sequence s' of common factors, and an integer $m > 1$, we can easily check in polynomial time whether s' consists of m or more factors and satisfy the length constraints if any, and whether the factors in s' appear in the right order in every sequence of S.

We next prove $MLCS \leq_P CMG$ which shows that CMG is NP-hard.

Given an instance of MLCS over an alphabet Σ given by sequences $T = \{T_1, T_2, \ldots, T_d\}$, we construct an instance of CMG, $S = \{S_1, S_2, \ldots, S_d\}$ over the alphabet $\{A, C, G, T\}$ such that there exists an LCS of length k for set T if, and only if, S has a CMG of k factors as follows:

1. First we relabel the characters of the MLCS instance using integers from $\{1, \ldots, |\Sigma|\}$ and convert each integer into its binary form.
2. We then replace '0's and '1's by 'A's and 'C's respectively to get strings over $\{A, C\}$ for each integer.
3. Finally we put these strings in the same order in each S_i as the corresponding integers appeared in T_i separated by 'G's in S_1 and 'T's in all other S_i and we set minimum factor length $p = \lceil \log |\Sigma| \rceil$.

Following is an example of the construction:

a b c d	1 2 3 4	001 010 011 100	AAC G ACA G ACC G CAA
b a c d $\Rightarrow$	2 1 3 4 $\Rightarrow$	010 001 011 100	$\Rightarrow$ ACA T AAC T ACC T CAA
a c b d	1 3 2 4	001 011 010 100	AAC T ACC T ACA T CAA

Now we show that this transformation of T into S is a reduction.

First suppose that T has a solution, that is a sequence t' of k characters are present in every sequence of T in exactly the same order. These characters, i.e., integers in t' will correspond to substrings of each S_i and these substrings will form a sequence s' of k factors. s' is a common motif sequence with k factors since according to the construction the factors must appear in each S_i in exactly the same order and there must be a gap of length at least one between any two factors.

Conversely, suppose, S has a common gapped motif sequence s' with k factors, $k > 1$, that follows a certain order in every sequence S_i. Note that since 'G' was used as the separator in S_1 and 'T' was used in all other strings, each factor must be strings over $\{A, C\}$ and corresponds to an integer in the LCS instance by construction, and they will follow the same order in every sequence giving us a common subsequence of length k in all strings in T. □

4 Algorithms

Algorithms are known for both MLCS and CMG problems that run in time polynomial in lengths of sequences and exponential in the number of sequences. Complexity results in [7] and in this paper indicate that asymptotically faster algorithms are unlikely. However, search space may be reduced by pruning leading to faster algorithms in practice. Here we present branch and bound algorithms for both MLCS and CMG problems.

4.1 A Branch and Bound Algorithm for MLCS Problem

Given a set of sequences, $S = \{S_1, S_2, \ldots, S_d\}$, where $|S_i| \leq n$ for $1 \leq i \leq d$, we preprocess the sequences to explore promising paths first and prune the search space using the value of the best solution found so far and the upper bound on values of solutions the path being explored may lead to. The algorithm uses memoization to avoid redundancy of work, i.e., the values of subproblems already calculated are stored in a table, V indexed by vectors of indices into the sequences.

Preprocessing. We preprocess the sequences to generate candidate lists that will be used to decide in what order nodes are visited during the search process. The candidate list for the first element of the longest common subsequence, C_{init}, consisting of triples $<element, minMultiplicity, minDistance>$, is generated as follows:

1. Process each sequence, S_i and list each element e, the distance of its first occurrence to the end of the sequence, $d_{e,i}$ and the number of times it appears in the sequence, $m_{e,i}$.
2. Intersect the lists to get elements common in all sequences. When we intersect we retain the minimum of distances to ends of the sequences for an element, and the minimum multiplicity of the element. Therefore, $minMultiplicity$ and $minDistance$ entries corresponding to $element$, which appears in all the sequences, are given by:

$$minMultiplicity = \min_{1 \leq i \leq d} m_{element,i}$$

$$minDistance = \min_{1 \leq i \leq d} d_{element,i}$$

3. Sort the triples in descending order of minimum distance to ends of sequences and record the sum of multiplicities.

The elements will be explored according to the order in the candidate list, the intuition being an element more distant to the ends of the sequences has more room for other elements to follow it in the LCS.

Similarly, for each element x that appears K_x times in all the sequences, we construct lists $C_{x,k}$ for $1 \leq k \leq K_x$ of triples corresponding to elements that follow the k-th occurrence of x in all the sequences.

For a finite alphabet, each such list can be constructed in time $O(nd)$ and since there can be at most n such lists, preprocessing takes $O(n^2 d)$ time.

Branch and Bound. At each node, we take as input a vector of indices $\mathbf{I} = <i_1, i_2, \ldots, i_d>$ and a common subsequence, α of the sequences $S_1[1 \ldots i_1], S_2[1 \ldots i_2], \ldots, S_d[1 \ldots i_d]$, i.e., common subsequence up to $\mathbf{I}$. We also maintain the best solution found so far globally. We start at $<0, \ldots, 0>$ with the common subsequence ϵ.

Then, at each node, we do the following:

1. Look up the last character and its multiplicity in α and retrieve the corresponding candidate list.
2. Iterate through the $<element, minMultiplicity, minDistance>$ triples in the candidate list.
3. Estimate upper bound (see **Pruning conditions** discussed shortly) to check if the branch can be pruned.

4. Find positions $\mathbf{P} = <p_1, p_2, \ldots, p_d>$ in each sequence following the input indices where *element* occurs. Note that such positions may not exist in some sequences as the k-th occurrence of x in α may correspond to a position in S_i to the right of the position of the k-th occurrence of x in S_i. We skip such entries.
5. If $<p_1, p_2, \ldots, p_d>$ has already been computed, then look up the value. Otherwise, explore $<p_1, p_2, \ldots, p_d>$ and update the best solution if needed.

Pruning Conditions. Suppose we are considering for exploration a triple, $<element, minMultiplicty, minDistance>$ and suppose this would be the ℓ-th occurrence of *element* in the common subsequence. The following properties can be used to calculate an upper bound on the maximum possible value, $\tilde{v}$ in the subtree rooted at the node:

1. $\tilde{v}$ can not exceed $1+minDistance$ since there are only $minDistance$ elements after *element* in at least one of the sequences.
2. Similarly, sum of multiplicities of $C_{element,\ell}$ is an upper bound on the number of elements that can follow *element* in the common subsequence.
3. Let $y = element$ and $p_i(y, \ell)$ be the position of the ℓ-th occurrence of y in the i-th sequence. Now $V[p_1(y, \ell), \ldots, p_d(y, \ell)]$ is an upper bound on $\tilde{v}$ because the position in S_i that corresponds to the ℓ-th occurrence of y in the common subsequence must be greater than or equal to $p_i(y, \ell)$ for $1 \leq i \leq d$.

The algorithm is summarized in Algorithm 1.

Algorithm 1. MLCS

```
1: Initialize: bestSolution ← 0
2: MLCS-B&B (<0, ..., 0>, ϵ)
3: procedure MLCS-B&B(I, α)
4:     x ← lastElement(α)
5:     k ← multiplicity(α, x)
6:     v ← 0
7:     for each <y, m, d> ∈ C_{k,x} do
8:         ℓ ← multiplicity(α, y)+1
9:         ṽ ← min(d + 1, multiplictySum(C_{y,ℓ}) + 1, V[p_1(y, ℓ), ... p_d(y, ℓ)])
10:        if ṽ + |α| > bestSolution then
11:            P ← getNextPos(I, y)
12:            if P is valid then
13:                if V[P] is not null then
14:                    v ← max(v, 1 + V[P])
15:                else
16:                    v ←max(v, 1+MLCS-B&B (P, α.y))
17:    if v + |α| > bestSolution then
18:        bestSolution ← v + |α|
19:    V[I] ← v
20:    Return v
```

An Illustrative Example. A simulation of the algorithm on an example is shown in Fig. 1.

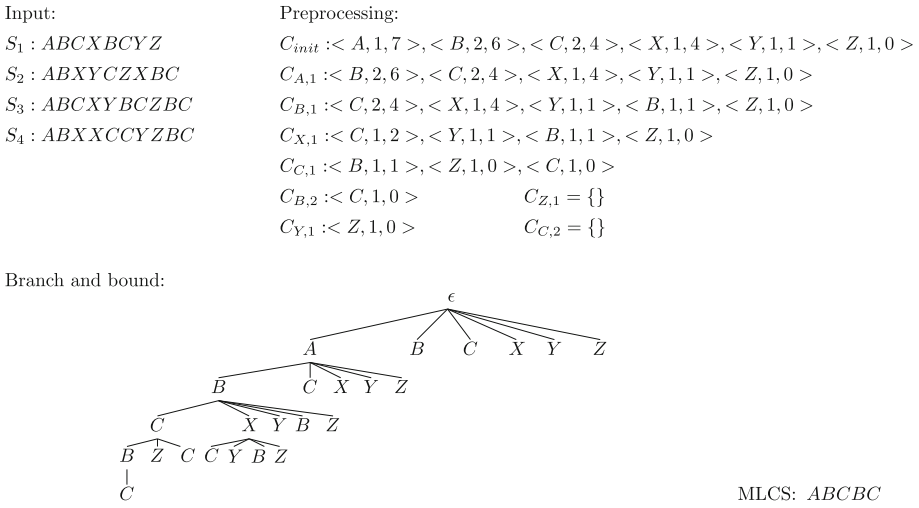

Fig. 1. Simulation of Algorithm 1 on an example set

4.2 A Branch and Bound Algorithm for CMG

We now extend the method discussed above and present a branch and bound algorithm for finding common motifs with gaps (CMG). We are given a set of strings, $S = \{S_1, S_2, \ldots, S_d\}$, and integers p, q giving upper and lower bounds on factor lengths respectively.

The first step is to find common factors. Then we use an approach similar to the one for finding MLCS taking into account that there may be multiple factors overlapping a position in a string, at most one of which can be present in the final solution.

Identifying Common Factors. We first identify common factors, i.e., substrings with lengths between p and q that appear in all the strings and record start indices (and end indices implicitly) of their occurrences in every string. This can be done efficiently using approaches such as suffix trees [8], finite automata [2]. For each string S_i, where $1 \leq i \leq d$, we create a list of factors, F_i consisting of all occurrences of all the common factors in the string sorted in ascending order of their end indices.

Candidate List Generation. The factor lists are then processed to generate candidate lists in a similar approach to the one used for preprocessing MLCS instances. In this context, a factor, f_1 will be in the list of candidates to follow

the k-th occurrence of factor f_2 if in each factor list there is an entry for f_1 with start index exceeding the end index of k-th occurrence of factor f_2 by at least 2. The candidate lists are sorted in descending order of minimum distances of factors from the ends of factor lists.

Branch and Bound and Pruning Conditions. The branch and bound algorithm now proceeds as the one for finding MLCS producing a common motif with highest number of factors.

An Illustrative Example. Figure 2 shows simulation of the algorithm for finding common motifs with gaps on a sample instance. The factor lists consist of pairs denoting the factor string and its start position sorted in increasing order of their end positions i.e. sum of the start positions and the lengths of factors. Note that although TCG follows TG in each of the factor lists, it is not included in the list of candidates to follow the first occurrence of TG since it overlaps with TG in two of the strings. We also see that although TGC is a common factor of the strings, it does not appear in the final common motif with gap as that would lead a motif with only one factor. However, substrings of TGC are considered as factors - TG in S_1 and S_3 and GC in S_2 - to obtain a common motif with three factors.

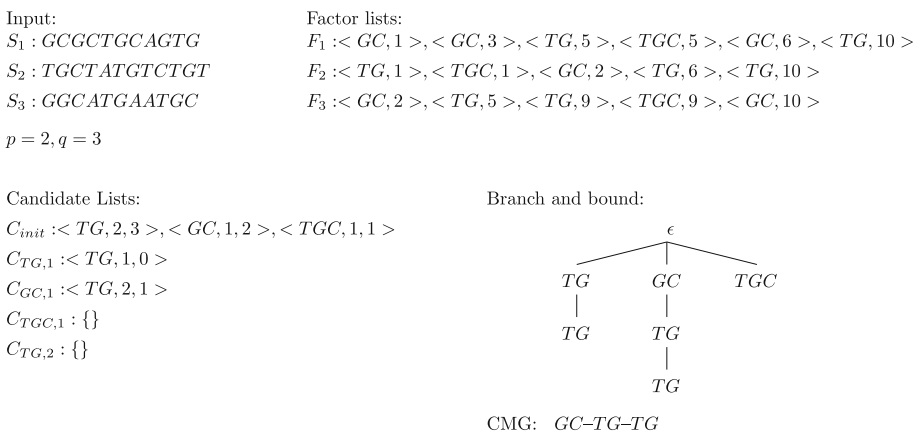

Fig. 2. Simulation of the algorithm for finding common motifs with gaps on an example

5 Conclusions

In this paper, we have addressed two problems with applications in genomics - finding a longest common subsequence of multiple sequences (MLCS) and finding common motifs with gaps (CMG). MLCS is known to be NP-hard and its reduction to CMG in this paper formally proves that CMG is NP-hard as well.

While this makes polynomial time algorithms for the problems unlikely, we have proposed a branch and bound algorithm with preprocessing to prune the search space that may reduce running time for many instances of the problems. Future work will include implementation of the algorithms to test the speed-up achieved compared to existing algorithms of the problems and subsequent application to real datasets.

References

1. Antoniou, P., Crochemore, M., Iliopoulos, C., Peterlongo, P.: Application of suffix trees for the acquisition of common motifs with gaps in a set of strings. In: International Conference on Language and Automata Theory and Applications (2007)
2. Antoniou, P., Holub, J., Iliopoulos, C.S., Melichar, B., Peterlongo, P.: Finding common motifs with gaps using finite automata. In: Ibarra, O.H., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 69–77. Springer, Heidelberg (2006). https://doi.org/10.1007/11812128_8
3. Chen, Y., Wan, A., Liu, W.: A fast parallel algorithm for finding the longest common sequence of multiple biosequences. BMC Bioinformatics **7**(4), S4 (2006)
4. Huang, G., Lim, A.: An effective branch-and-bound algorithm to solve the k-longest common subsequence problem. In: Proceedings of the 16th European Conference on Artificial Intelligence, pp. 191–195. IOS Press (2004)
5. Iliopoulos, C.S., McHugh, J., Peterlongo, P., Pisanti, N., Rytter, W., Sagot, M.F.: A first approach to finding common motifs with gaps. Int. J. Found. Comput. Sci. **16**(06), 1145–1154 (2005)
6. Korkin, D., Wang, Q., Shang, Y.: An efficient parallel algorithm for the multiple longest common subsequence (MLCS) problem. In: 2008 37th International Conference on Parallel Processing, ICPP 2008, pp. 354–363. IEEE (2008)
7. Maier, D.: The complexity of some problems on subsequences and supersequences. J. ACM (JACM) **25**(2), 322–336 (1978)
8. Marsan, L., Sagot, M.F.: Extracting structured motifs using a suffix tree algorithms and application to promoter consensus identification. In: Proceedings of the Fourth Annual International Conference on Computational Molecular Biology, pp. 210–219. ACM (2000)
9. Wang, Q., Korkin, D., Shang, Y.: A fast multiple longest common subsequence (MLCS) algorithm. IEEE Trans. Knowl. Data Eng. **23**(3), 321–334 (2011)
10. Yang, J., Xu, Y., Sun, G., Shang, Y.: A new progressive algorithm for a multiple longest common subsequences problem and its efficient parallelization. IEEE Trans. Parallel Distrib. Syst. **24**(5), 862–870 (2013)