Finding Common Motifs with Gaps using Finite Automata*

Pavlos Antoniou[1], Jan Holub[2], Costas S. Iliopoulos[1], Bořivoj Melichar[2], Pierre Peterlongo[3]

[1] Dept. of Computer Science, King's College London, London WC2R 2LS, England, UK, e-mail: {pavlos.antoniou, csi}@kcl.ac.uk
[2] Czech Technical University in Prague, Department of Computer Science and Engineering, Karlovo nám. 13, 121 35, Prague 2, Czech Republic, e-mail: {holub, melichar}@fel.cvut.cz
[3] Institut Gaspard-Monge, Université de Marne-la-Vallée, Cite Descartes, Champs sur Marne, 77454 Marne-la-Vallée CEDEX 2, France, e-mail: pierre.peterlongo@univ-mlv.fr

Abstract. We present an algorithm that uses finite automata to find the common motifs with gaps occurring in all strings belonging to a finite set $S = \{S_1, S_2, \ldots, S_r\}$. In order to find these common motifs we must first identify the factors that exist in each string. Therefore the algorithm begins by constructing a factor automaton for each string S_i. To find the common factors of all the strings, the algorithm needs to gather all the factors from the strings together in one data structure and this is achieved by computing an automaton that accepts the union of the above-mentioned automata. Using this automaton we are able to create a new factor alphabet. Based on this factor alphabet a finite automaton is created for each string S_i that accepts sequences of all non overlapping factors residing in each string. The intersection of the latter automata produces the finite automaton which accepts all the common subsequences with gaps over the factor alphabet that are present in all the strings of the set $S = \{S_1, S_2, \ldots, S_r\}$. These common subsequences are the common motifs of the strings.

1 Introduction

The problem of finding common motifs in a set of strings has long been an area of interest in the academic community. Given a set of strings, the problem of finding common motifs in that set is the problem of finding similar substrings that lie in all of these strings. In some particular applications, like in biology, this requirement is more flexible in the sense that

* This research has been partially supported by the Ministry of Education, Youth and Sports under research program MSM 6840770014 and the Czech Science Foundation as project No. 201/06/1039.

motifs do not have to be identical but have to share a certain degree of similarity. This degree is quantified using metrics such as Hamming and Levenshtein distances or by allowing don't care symbols to occur in the motifs. Don't care symbols are occurrences in the string that can match any symbol of the alphabet. In this paper, we are interested in finding common motifs in the strings that have don't care symbols concentrated in distinct parts of contiguous positions in the strings, i.e. common motifs with gaps.

This problem has engrossed biologists because of its applications in that area. It can be applied in understanding the fundamental process of gene expression [8]. Gene expression consists of two parts, transcription and translation. During transcription an mRNA molecule is created by copying a gene from the DNA and during translation the mRNA molecule is decoded to produce a protein. In order though for the transcription process to begin, one or more proteins, called transcription factors, have to bind to some specific regions of the gene called binding sites. These binding sites share common patterns which are the common motifs of the genes. If these common motifs are identified and extracted from the genes, they will give the opportunity to biologists to match these binding sites to their corresponding transcription factors in order to be able to fully understand the way gene expression works [8].

A classical approach to finding these motifs was by using artificial intelligence techniques [10] but these methods are inexact methods that used machine learning to discover the motifs by training the machines to recognize them. Recently, microarray technology has been used particularly in this application of the problem but this technology is inexact, it is based on probabilities and is limited by weak signal sequences [8].

In text algorithm applications, finding common motifs with gaps has been mainly handled using suffix trees [1, 2, 5, 7] which provided exact results. In this paper we propose an algorithm using automata to index common gapped motifs. We believe that the use of automaton permit the indexation of bigger strings and allows more open definitions.

Section 2 formally introduces the general problem. Section 3 presents an algorithm in order to solve the question of finding common motifs with gaps. Moreover Section 4 presents a complete example following step by step the proposed algorithm. Eventually in Section 5 there is an analysis of the complexity of the proposed solution.

2 Definition of the problem

Given a set of strings $S = \{S_1, S_2, \ldots, S_r\}$ and p, q, $1 \leq p \leq q \leq \min(|S_j| : j \in \langle 1, r \rangle)$. The problem of finding common motifs with gaps consists in finding words $B_1, B_2, \ldots, B_m$ such that:

1. $m > 1$.
2. $p \leq |B_i| \leq q$ for $i \in \langle 1, m \rangle$.
3. $B_1 \circ^{d_{i,1}} B_2 \circ^{d_{i,2}} \ldots \circ^{d_{i,m-1}} B_m$ occur in S_i for all $i \in \langle 1, r \rangle$, $m > 1$ and the size of the gap $d_{i,j}$ varies in each motif (Fig. 1), where $\circ$ denotes don't care symbol matching any symbol of alphabet and $\circ^j$ denotes concatenation of j don't care symbols.

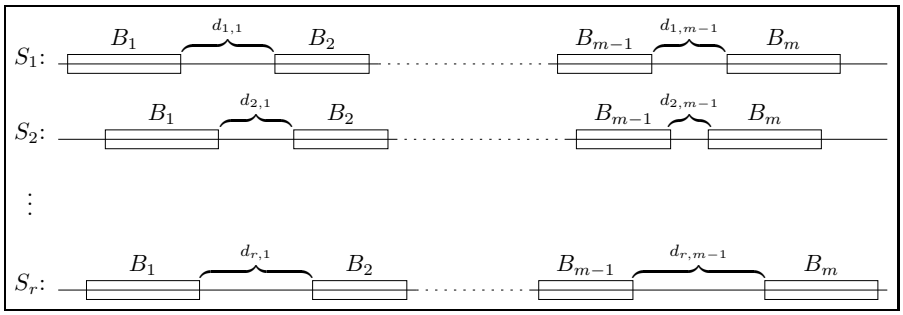

Fig. 1. An example of a motif with gaps that occurs in every string, where by $d_{i,j}$ we mean a size of a gap

3 Algorithm

The algorithm takes as input a set of strings $S = \{S_1, S_2, \ldots, S_r\}$ and two constants p, q, which will be the lower and upper bound respectively of the length each motif can have, and returns the common motifs with gaps found in all those strings. The algorithm begins with computing the set of all factors F of length between the constants p and q that appear in all strings belonging to the set $S = \{S_1, S_2, \ldots, S_r\}$.

In order to find all these factors in F that appear in all the strings, we begin by creating a factor automaton MF_i for each string $S_i \in S$. Each factor automaton MF_i accepts all the factors of the particular string $S_i \in S$. Then, the algorithm joins all the MF_i automata together in one

automaton. The resulting union automaton accepts the union of the languages accepted by each of the r automata. This automaton can either be deterministic or non-deterministic. If it is deterministic, then the algorithm finishes because this is a sign that there are no common symbols and therefore no common motifs are present in the strings from this set S. On the other hand, if the resulting union automaton is non-deterministic, the algorithm proceeds with transforming this non-deterministic automaton into a deterministic one.

From this deterministic union automaton we identify all factors having length between the two constants p and q that are repeating in all strings from the set S. These factors are subsequently used to create a repetition table RT, which is used to create a new factor alphabet containing only the symbols relevant to the factors extracted in the previous steps.

Based on the repetition table, we find the longest common subsequence over the factor alphabet of all the strings of S. To achieve this aim, we first create a finite automaton MS_i for each string $S_i \in S$ accepting sequences of non-overlapping factors using the factor alphabet as input alphabet. Then, we create the automaton MS by taking the intersection of these automata MS_i. The resulting automaton will accept the intersection of the languages accepted by each of the factor automata i.e. it accepts all sequences of factors occurring in all strings from the set S which are the common motifs of the strings with gaps.

The algorithm:
Input: Set of strings $S = \{S_1, S_2, \ldots, S_r\},\ p,\ q$.
Output: Sequence of words $B_1, B_2, \ldots, B_m$ occurring in all strings in S.
Method:

1. (a) For each string $S_i \in S$ construct a factor automaton $M_{i\varepsilon}$ by creating automaton M_i, accepting string S_i (i.e. $L(M_i) = \{S_i\}$), then adding ε-transitions leading from the initial state to all states of M_i and making all states final.
   (b) Construct automaton M_ε, $L(M_\varepsilon) = \bigcup_{i=1}^{r} L(M_{i\varepsilon})$.
   (c) By eliminating ε-transitions in M_ε we get M_F.
   (d) If M_F is deterministic, then strings in S have no common symbol and thus they cannot have a common motif. Set $m = 0$ and exit the algorithm.
   (e) Using determinisation of M_F we construct M_{DF} while for each state q' of M_{DF} we preserve a set of states of M_F q' consists of. The set is called d-subset.

2. Find all states of M_{DF} representing factors of length between p and q and having at least one state from each automaton M_i in its d-subset. Construct a repetition table RT (the shortest path from the initial state to the state spells the repeated factor while members of d-subset identify locations).
3. Take all factors represented by states in the previous step and create a new "factor alphabet" *FA*.
4. For each string S_i in S construct a finite automaton MS_i accepting sequences of all non-overlapping factors from *FA*.
5. Construct automaton *MS* accepting all common subsequences of sequences accepted by automata MS_i for $i \in \langle 1, r\rangle$ using the following approach:
   (a) Add ε-transitions parallel to each transition in each finite automaton $MS_i, i \in \langle 1, r\rangle$. The resulting automata will be MS_i^{ε}.
   (b) By eliminating ε-transitions in MS_i^{ε} we get MS_i^N for each $i \in \langle 1, r\rangle$.
   (c) Construct the automaton *MS*, $L(MS) = \bigcap_{i=1}^{r} L(MS_i^N)$.
   (d) Finite automaton *MS* is accepting all sequences $B_1, B_2, \ldots, B_m$ which are sequences of factors occurring in all strings from set S.

4 An example

As an example let's consider a set of strings $S = \{aabccddab, babbcdacd\}$. We will find common motifs in this set of strings bounded from parameters $p = 2, q = 3$.

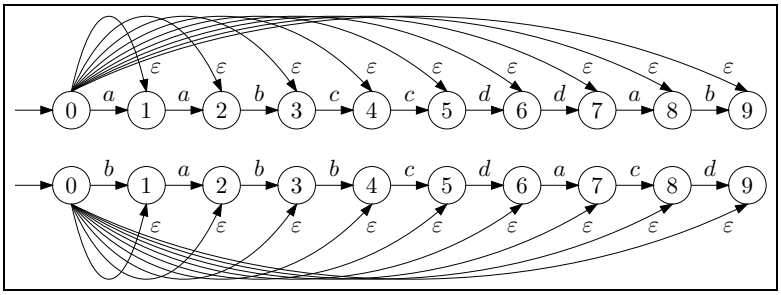

Fig. 2. Transition diagrams of finite automata $M_{1\varepsilon}$ and $M_{2\varepsilon}$ for the set of strings $S = \{aabccddab, babbcdacd\}$ from the example

First (step 1a of the algorithm) we construct finite automata $M_{1\varepsilon}$ and $M_{2\varepsilon}$ for both strings from S. See Fig. 2[1].

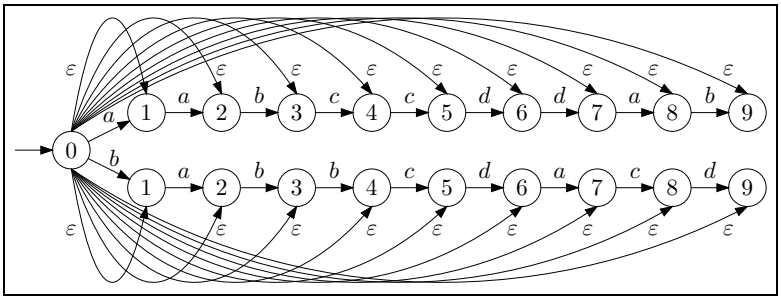

Fig. 3. Transition diagram of finite automaton M_ε from the example

In the next step (step 1b of the algorithm) we construct automaton M_ε accepting the union of languages $L(M_{1\varepsilon})$ and $L(M_{2\varepsilon})$. See Fig. 3.

According to step 1c of the algorithm, we construct automaton M_F by replacing the ε-transitions by non ε-transitions. The transition diagram of the resulting automaton is given in Fig. 4.

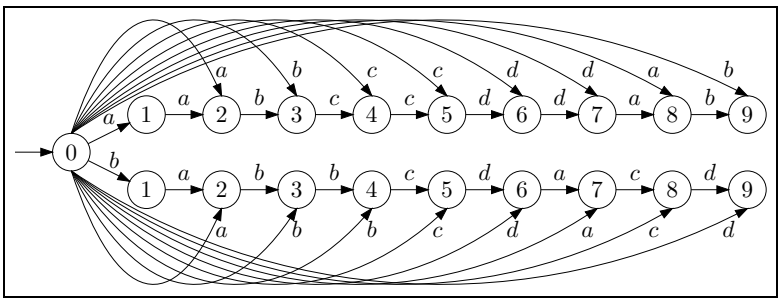

Fig. 4. Transition diagram of finite automaton M_F from the example

In this example, automaton M_F is nondeterministic. This means that there is a possibility that a common motif exists in set S. According to step 1e of the algorithm 3, we must construct its deterministic equivalent M_D. Its transition diagram is given in Fig. 5.

[1] All states in the automata presented in this paper are final states.

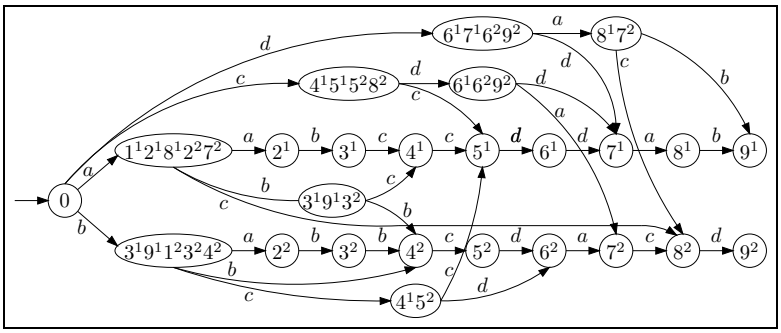

Fig. 5. Transition diagram of finite deterministic factor automaton M_D from the example

Factor	d-subset	Repetitions
ab	$3^1 9^1 3^2$	$(3^1, F), (9^1, G), (3^2, F)$
bc	$4^1 5^2$	$(4^1, F), (5^2, F)$
cd	$6^1 6^2 9^2$	$(6^1, F), (6^2, F), (9^2, G)$
da	$8^1 7^2$	$(8^1, F), (7^2, F)$

Table 1. Repetition table RT of common factors from the example (F – first occurrence, G – repetition with a gap)

Table 1 is the repetition table RT of common factors created as described in step 2 of the algorithm. The factor alphabet (step 3 of the algorithm) is $FA = \{ab, bc, cd, da\}$. Subsequently we will construct, according to step 4 of the algorithm, finite automata MS_1 and MS_2 accepting all non-overlapping sequences of factors of the both strings. Their transition diagrams are depicted in Fig. 6.

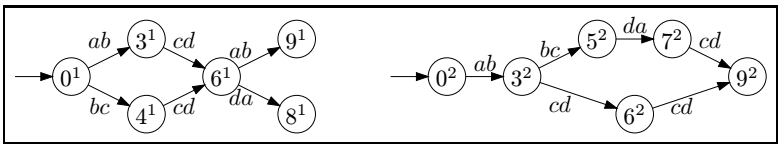

Fig. 6. Transition diagrams of finite automata MS_1 and MS_2 accepting sequences of non-overlapping factors of both strings from the example

The last step (step 5 of the algorithm involves the construction of an automaton which accepts all sequences of factors occurring in both

strings of S. Transition diagrams of finite automata MS_1^{ε} and MS_2^{ε} are shown in Fig. 7 (step 5a).

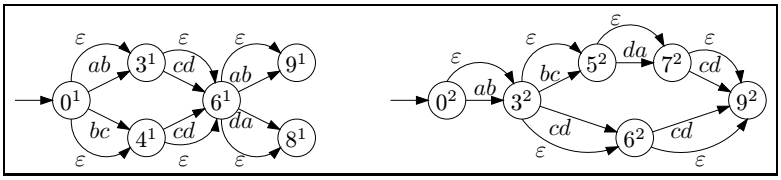

Fig. 7. Transition diagrams of finite automata MS_1^{ε} and MS_2^{ε} from the example

Transition diagrams of finite automata MS_1^N and MS_2^N are shown in Fig. 8 (step 5b of the algorithm).

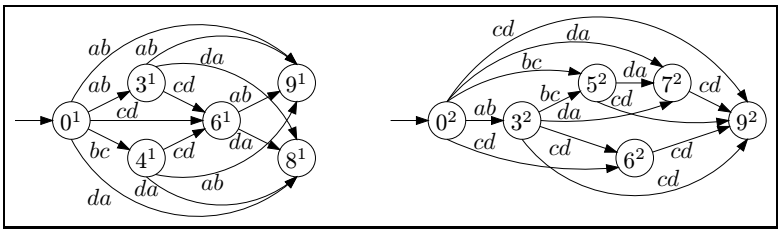

Fig. 8. Transition diagrams of finite automaton MS_1^N and MS_2^N from the example

According to the step 5c of the algorithm, we need to construct a finite automaton accepting the language that corresponds to the intersection of the languages accepted by the two automata. The transition diagrams of the finite automaton accepting the intersection of the languages accepted by automaton MS_1^N and MS_2^N is depicted in Fig 9.

Finally, from the above computation we may conclude that in this particular set $S = \{aabccddab, babbcdacd\}$ and using parameters $p = 2, q = 3$ the following common motifs occur: $m_1 = \{ab, cd\}$, $m_2 = \{ab, da\}$, $m_3 = \{bc, cd\}$, $m_4 = \{bc, da\}$.

5 Time and Space Complexity of the algorithm

We shall discuss the time and space complexity for each step of the algorithm. As described in Section 3 the algorithm requires five steps.

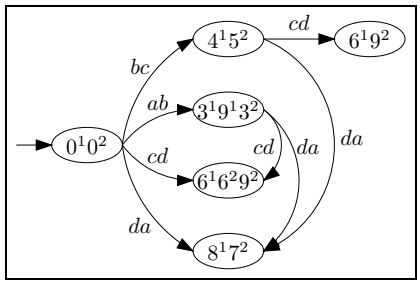

Fig. 9. Transition diagram of finite automaton *MS* from the example

In Step 1 we fist construct r finite automata, one for each string S_i of the set of strings S. The time and space needed to construct each automaton depends on the length of each particular string S_i which is the language to be accepted by the automaton. Therefore this process requires linear time and space [4] with respect to the length of the strings.

Subsequently we wish to construct the automaton which accepts the union of these languages. If we assume that the value n is the cumulated size of the input sequences i.e. $n = \sum_{i=1}^{r} |S_i|$, then this step requires $\mathcal{O}(n)$ space and time for all the strings to be included in the automaton. Next, we wish to transform this union automaton into a deterministic factor automaton also called Direct Acyclic Word Graph (DAWG). In most cases the union automaton is non-deterministic. In order to create the deterministic factor automaton we need to transform the non-deterministic union automaton to a deterministic automaton which will be the factor automaton M_F. Generally, the construction of a deterministic automaton from a non-deterministic requires exponential time and space. In the case of factor automata though the maximum number of states of the resulting deterministic automaton is $2|n| - 1$ and the maximum number of transitions is $3|n| - 4$ [3]. Thus, this step is bounded overall by linear time and space.

In Step 2 we extract from M_F the factors belonging to all the strings and which have length between the values p and q, and we add them to the repetition table RT. In order to find all these factors we need to reach all the states at depth q. Let $\delta = q - p + 1$ the length of the last interval. As we are looking for all the factors between p and q there are at most $n \times \delta$ such factors. Thus the complexity of this step is $\mathcal{O}(n \times \delta)$.

In Steps 3 and 4 we construct the new factor alphabet and for each string S_i in S we construct a finite automaton MS_i accepting all non-overlapping factors from the factor alphabet. Each automaton requires $\mathcal{O}(n)$ time to be constructed and $\mathcal{O}(n)$ space. Thus, overall this step can be completed in linear time and requires linear space.

In Step 5 we construct the finite automaton MS accepting all common subsequences of the strings accepted by MS_i. This is achieved by creating the finite automaton MS that accepts the intersection of the languages accepted by automata MS_i taken from Step 4. The process of intersecting automata requires quadratic time and is usually done by cartesian product. In [6], Holub and Melichar present an algorithm for the intersection of factor automata which does not employ cartesian product but uses state marking. Using this algorithm we avoid the creation of all inaccessible states during the automaton construction.

Although the resulting automaton from the algorithm in [6] contains no inaccessible states as it would have if we had used cartesian product to construct it, nevertheless the time and space complexities of this step are still quadratic relative to the input. For the case of only two factor automata to be intersected, for example L_1 and L_2 having lengths n and m respectively then the state complexity of $L_1 \cap L_2$ is $\mathcal{O}(nm)$. When we transfer this into a problem with many automata the complexity will become $\mathcal{O}(n^k)$—polynomial with n size of texts and exponential with k the number of automata. This is a familiar situation relating to the problem of finding the longest common subsequences of many (> 3) strings using Dynamic Programming which is an NP-complete problem so no better exact algorithm is destined to appear [9].

Overall, looking over all the steps we can see that the algorithm's time and space complexity is exponential due to the last step that requires the intersection of many finite automata.

Conclusion

We have presented a complete automaton based algorithm to solve the problem of identifying and indexing the common motifs with gaps in a set of strings. The algorithm takes advantage of the fact that one can find common motifs of a set of strings by intersecting their corresponding factor automata which were created by the common factors residing in the strings. Other solutions of the problem require some limit of gaps (fixed gap, bounded gap, bounded sum of gaps). The presented algorithm allows any gaps while keeping the same time and space complexity. Moreover it

offers a sound application of finite automata on the problem of finding common motifs with gaps in a set of strings.

References

1. C. Charras and T. Lecroq. *Exact string matching algorithms.* 2004.
2. T. Crawford, C. S. Iliopoulos, and R. Raman. String matching techniques for musical similarity and melodic recognition. *Computing in Musicology*, 11:73–100, 1998.
3. M. Crochemore and R. Vérin. Direct construction of compact directed acyclic word graphs. In A. Apostolico and J. Hein, editors, *Proceedings of the 8th Annual Symposium on Combinatorial Pattern Matching*, number 1264, pages 116–129, Aarhus, Denmark, 1997. Springer-Verlag, Berlin.
4. Maxime Crochemore and Christophe Hancart. Automata for matching patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 2, Linear Modeling: Background and Application, chapter 9, pages 399–462. Springer-Verlag, 1997.
5. Maxime Crochemore and Wojciech Rytter. *Text algorithms.* Oxford University Press, Inc., New York, NY, USA, 1994.
6. Jan Holub and Bořivoj Melichar. Approximate string matching using factor automata. *Theor. Comput. Sci.*, 249(2):305–311, 2000.
7. C. S. Iliopoulos, J. McHugh, P. Peterlongo, N. Pisanti, W. Rytter, and M. Sagot. A first approach to finding common motifs with gaps. *International Journal of Foundations of Computer Science*, 2004.
8. Henry C.M. Leung. Finding motifs with insufficient number of strong binding sites. *Journal of Computational Biology*, 12(6):686–701, 2005.
9. Steven S. Skiena. *The algorithm design manual.* Springer-Verlag New York, Inc., New York, NY, USA, 1998.
10. Michael E. Baker Timothy L. Bailey and Charles P. Elkan. An artificial intelligence approach to motif discovery in protein sequences: Application to steroid dehydrogenases. *The Journal of Steroid Biochemistry and Molecular Biology*, 62(1):29–44, 1997.