Composite Pattern Discovery for PCR Application

Stanislav Angelov[1⋆] and Shunsuke Inenaga[2]

[1] Department of Computer and Information Science,
School of Engineering and Applied Sciences,
University of Pennsylvania, Philadelphia, PA 19104, USA
angelov@cis.upenn.edu
[2] Department of Informatics, Kyushu University, Fukuoka 812-8581, Japan
shunsuke.inenaga@i.kyushu-u.ac.jp

Abstract. We consider the problem of finding pairs of short patterns such that, in a given input sequence of length n, the distance between each pair's patterns is at least α. The problem was introduced in [1] and is motivated by the optimization of multiplexed nested PCR.
We study algorithms for the following two cases; the special case when the two patterns in the pair are required to have the same length, and the more general case when the patterns can have different lengths. For the first case we present an $O(\alpha n \log\log n)$ time and $O(n)$ space algorithm, and for the general case we give an $O(\alpha n \log n)$ time and $O(n)$ space algorithm. The algorithms work for any alphabet size and use asymptotically less space than the algorithms presented in [1]. For alphabets of constant size we also give an $O(n\sqrt{n}\log^2 n)$ time algorithm for the general case. We demonstrate that the algorithms perform well in practice and present our findings for the human genome.
In addition, we study an extended version of the problem where patterns in the pair occur at certain positions at a distance at most α, but do not occur α-close anywhere else, in the input sequence.

1 Introduction

1.1 Composite Pattern Discovery

Pattern discovery is a fundamental problem in Computational Biology and Bioinformatics [2, 3]. A large amount of effort was paid to devising efficient algorithms to extract interesting, useful, and surprising substring patterns from massive biological sequences [4, 5]. Then this research has been extended to more complicated but very expressive pattern classes such as subsequence patterns [6, 7], episode patterns [8, 9], VLDC patterns [10], and their variations [11].

The demand for *composite pattern discovery* has recently arisen rather than simply finding single patterns. It is motivated by, for instance, the fact that many

⋆ Supported in part by NSF Career Award CCR-0093117, NSF Award ITR 0205456 and NIGMS Award 1-P20-GM-6912-1.

of the actual regulatory signals are composite patterns that are groups of monad patterns occurring near each other [12]. The concept of composite patterns was introduced by Marsan and Sagot [13] as *structured motifs* which are two or more patterns separated by a certain distance. They introduced suffix tree [14] based algorithms for finding structured motifs, and Carvalho et al. [15] presented a new algorithm with improved running time and space.

In a similar concept, Arimura et al. [16, 17] introduced *proximity patterns* and proposed algorithms to find these patterns efficiently. MITRA [12] is another method that looks for composite patterns. BioProspector [18] applies the Gibbs sampling strategy to discover gapped motifs. Bannai et al. [19] and Inenaga et al. [20] considered *Boolean combinations* of patterns, in order to find regulatory elements that cooperate, complement, or compete with each other in enhancing and/or silencing certain genomic functions.

Another application of composite pattern discovery is to find good adapters for primers used in *polymerase chain reaction* (*PCR*). PCR is a standard technique for producing many copies of a DNA region [21]. It is routinely used for example in medicine to detect infections and in forensic science to identify individuals even from tiny samples. In PCR a pair of short fragments of DNA called primers is specifically designed for the amplified region so that each of them is complementary to the 3' end of one of two strands of the region (see also Fig. 1).

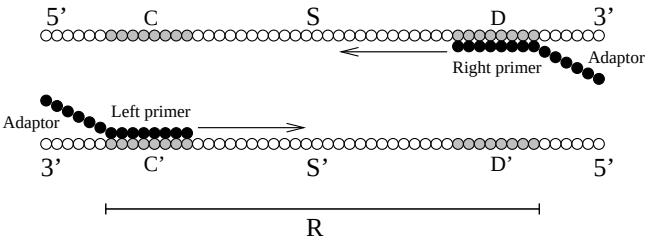

Fig. 1. Illustration for polymerase chain reaction (PCR).

In order to achieve ultrasensitive detection, repeated PCR with nested primers, so-called *nested PCR*, is used. Also, detection tests are preferred to be carried out in a multiplexed fashion. Let S denote any sequence taken from a sample of genome, and S' denote the reverse complement of S. To obtain a good primer pair for multiplexed nested PCR, we are required to find a pair of patterns (A, B) such that any occurrences of A and B are separated further than α in both sequences S and S', where α is a given threshold value. Then, the pair (A, B) is called a *missing pattern pair* (or shortly a *missing pair*). For the application of multiplexed nested PCR, the patterns in a missing pair have to also satisfy that $|A| = |B| = k$ and k is as short as possible. Namely, a missing pair with patterns of the same, and shortest length, is demanded. More details of the relationship between missing patterns and PCR can be found in [1].

Table 1. Summary of results for finding missing pairs of patterns of same length.

Algorithm	Time	Space
Algorithm 1	$O(\alpha n \log\log_\sigma n)$	$O(n)$
Bit Table Algorithm [1]	$O(\alpha n(\sigma + \log\log_\sigma n))$	$O(\alpha n)$

1.2 Finding Missing Patterns

The problem of finding missing pattern pairs was firstly considered in [1]. The paper presented an algorithm which finds missing pattern pairs in $O(\alpha n \log\log n)$ time with $O(\alpha n)$ space, where n denotes the length of the input sequence. For a more general case where the two patterns in a missing pair can have different lengths, the paper showed that the problem is solvable in $O(n^2)$ time and $O(n)$ space, or in $O(\alpha n \log n)$ time and $O(n \log n)$ space, both on a constant-size alphabet. We remark that the patterns considered in [1] were substring patterns, that is, exact match without errors was considered.

In this paper, we give simpler and more efficient algorithms that solve the stated problems for an arbitrary alphabet size σ. We give an $O(\alpha n \log\log_\sigma n)$ time algorithm for the case when the patterns in the missing pairs are of the same length, and $O(\min\{\alpha n \log_\sigma n, (\sigma + \log n)n\sqrt{n}\log_\sigma n\})$ time algorithm for the case when the two patterns can have different lengths. In both cases the space requirement is only $O(n)$.

See Tables 1 and 2 for more detailed comparison between our algorithms and those obtain in [1]. For patterns of the same length and constant-size alphabets, Algorithm 1 saves computational space by a factor of α. It also improves the time complexity for arbitrary alphabet sizes. For pairs of patterns of different lengths, Algorithm 1 is superior to Suffix Tree Algorithm B on both constant and arbitrary alphabets. It is also noteworthy that although both Suffix Tree Algorithms heavily depend on manipulations on suffix trees [14], neither Algorithm 1 or 2 in this paper needs advanced data structures which can be rather expensive in practice.

Furthermore, since primers need to occur around the region to be amplified, we also study a natural extension of the problem where patterns in the pair occur at certain positions at a distance at most α, but do not occur α-close anywhere else, in the input sequence. We show how Algorithm 1 can be modified for this extended problem. Since the restriction can make "short" pattern pairs impossible, we also discuss a variant that allows for arbitrary pattern lengths. We note that for the case of primers, which typically have lengths in the range 17..25, the obtained algorithm runs in $O(\alpha n)$ time and $O(n)$ space.

1.3 Organization

In Section 2 we formally introduce our model and state the considered problem. In Section 3 we present the main algorithm, and subsequent results. Next, in

Table 2. Summary of results for finding missing pairs of patterns of different length.

Algorithm	Time	Space
Algorithm 1	$O(\alpha n \log_\sigma n)$	$O(n)$
Algorithm 2	$O((\sigma + \log n) n\sqrt{n} \log_\sigma n)$	$O(n)$
Suffix Tree Alg. A [1]	$O(n^2)$	$O(n)$
Suffix Tree Alg. B [1]*	$O(\alpha n \log n)$	$O(n \log n)$
Suffix Tree Alg. B [1]	$O(\log \sigma \alpha n \log_\sigma n)$	$O(\log \sigma \alpha n \log_\sigma n)$

* Constant size alphabet.

Section 4, we discuss natural extensions to the main algorithm and their implications. In Section 5 we discuss our findings on the human genome. Finally, Section 6 concludes this paper with some possible directions for future work.

2 Preliminaries

A *string* $T = t_1 t_2 \cdots t_n$ is a sequence of *characters* from an ordered *alphabet* Σ of size σ. We assume *w.l.o.g.* $\Sigma = \{0, 1, \ldots, \sigma - 1\}$ and that all characters occur in T. A *substring* of T is any string $T_{i...j} = t_i t_{i+1} \cdots t_j$, where $1 \leq i \leq j \leq n$. A *pattern* is a short string over the alphabet Σ. We say that pattern $P = p_1 p_2 \cdots p_k$ *occurs* at position j of string T iff $p_1 = t_j, p_2 = t_{j+1}, \ldots, p_k = t_{j+k-1}$. Such positions j are called the occurrence positions of P in T.

A *missing pattern* P (with respect to sequence T) is such that P is not a substring of T, i.e., P does not occur at any position j of T. Let $\alpha > 0$ be a threshold parameter. A *missing pattern pair* (A, B) is such that if A (resp. B) occurs at position j of sequence T, then B (A) does not occur at any position j' of T, such that $j - \alpha \leq j' \leq j + \alpha$. If (A, B) is a missing pair, we say that A and B do not occur α-*close* in T. These notions are illustrated in Fig. 2.

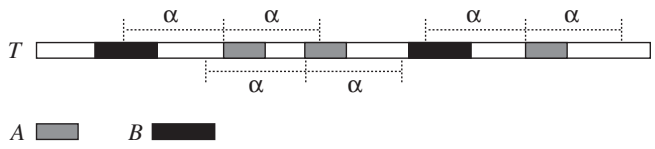

Fig. 2. Missing pattern pair (A, B). No occurrences of A and B are at a distance closer than α.

We study the following problem:

Problem 1 (Missing Pattern Pair Problem). Given a sequence T and a threshold α, find patterns A and B of minimum total length, such that (A, B) is a missing pattern pair with respect to T, i.e., A and B do not occur α-close in T.

3 Finding Missing Pattern Pairs

Missing pattern pairs can be formed by two processes. When a pattern does not occur in the input sequence T, it can be combined with any pattern to form a missing pair. Alternatively, both patterns in the pair may occur in the sequence, but always at least α positions away from each other. The first case, when a single pattern is missing, provides an insight to the upper bound on the missing pair length and is an interesting property on its own.

It is not hard to see that there is a missing pattern of length $\lceil \log_\sigma n \rceil$ from sequence T with size n, where σ is the input alphabet size. This is because there are at most $n - k + 1$ distinct patterns of length k in T. In [1], a linear time algorithm based on suffix trees is proposed that finds the shortest missing pattern when σ is a constant. The algorithm can be readily extended for the case of arbitrary alphabet sizes by a loss of $\log \sigma$ factor. Instead, we can compute a bit table of all patterns of length $\lfloor \log_\sigma n \rfloor$ that occur in T using the natural bijective mapping of the patterns to the integers $0, 1, \ldots, \sigma^{\lfloor \log_\sigma n \rfloor} - 1$. This can be done in linear time by scanning the input sequence from left to right using the established technique of computing the entry of pattern Yb knowing the entry of pattern aY (see for example [22]). By examining consecutive runs of missing patterns of this length, one can compute the shortest string (the longest missing pattern prefix) that is missing from the input sequence. If all patterns of length $\lfloor \log_\sigma n \rfloor$ are present in T, then the shortest missing pattern is of length $\lceil \log_\sigma n \rceil$. In this case we can find a representative by computing the first n entries of the corresponding bit table.

Proposition 1. *The shortest* single *missing pattern problem can be solved in linear time and space.*

In what follows, we let m be the length of the shortest missing pattern.

3.1 Finding Missing Pairs of Fixed Lengths

We now present an $O(\alpha n)$ time and $O(n)$ space algorithm that finds a missing pattern pair (A, B), where the lengths of A and B are given as input parameters. The algorithm serves as a basis for the missing pattern pairs algorithms that follow. Let $|A| = a$ and $|B| = b$ and assume *w.l.o.g.* $a \geq b$. We will first consider the case when $a < m$, or else there is a pattern of length a that is missing and by Proposition 1 it can be found in linear time. Let $N_1 = \sigma^a$ and $N_2 = \sigma^b$ be the number of distinct patterns of length a and b respectively. (Clearly, $n > N_1 \geq N_2$). The proposed algorithm heavily uses the bijective mapping of patterns to integers described in the previous subsection.

Algorithm 1. We now describe the steps of the algorithm.

1. Let L be an array of length N_1, where $L[h]$ is the list of occurrence positions in T of the pattern of length a mapped to the integer h.

2. Compute an array H s.t. $H[j]$ is the mapped value of the pattern of length b at position j of T.
3. For $h = 0 \dots N_1 - 1$, count the number of distinct patterns B of length b that occur at distance at most α from the pattern A of length a mapped to h. We do this by maintaining a bit table of the distinct patterns B that are α-close to A.

   At each iteration we perform the following sub-steps. Let A be the pattern mapped to h.
   (i) For each occurrence in $L[h]$ of pattern A, we mark in a table M of size N_2 all patterns of length b that occur at distance at most α by scanning the corresponding positions of the array H.
   (ii) When a pattern of length b is seen for the first time we increment a counter. The counter is set to 0 at the beginning of each iteration.
   (iii) The iteration ends when the maintained counter becomes equal to N_2 to indicate that all patterns of length b are α-close to A, or when all of $L[h]$ is processed. At the end of an iteration, if the counter is less than N_2 we scan M to output a missing pattern pair and the algorithm terminates.

Analysis. Step 1 of the algorithm can be performed in $O(n)$ time by scanning T from left to right. Compute the value h of the pattern at position i from that of position $i-1$ and append i to the list $L[h]$. The total size of all lists is $n-a+1$. The array H in Step 2 can be computed in a similar fashion and takes $n-b+1$ space. An iteration of Step 3 takes $O(\alpha|L[h]|)$ time for a total of $O(\alpha n)$ time and an additional $O(N_2) = O(n)$ space. We conclude the algorithm will output a missing pair (A, B) with the desired pattern lengths, if such pair exists, in $O(\alpha n)$ time and $O(n)$ space.

3.2 Finding Missing Pairs of the Same Length

We combine the algorithm from the previous subsection and the following property to obtain an efficient algorithm for the problem of finding missing pairs when the patterns are of the same length.

Property 1 (Monotonicity Property). If a pattern pair (A, B) is missing, the pair (C, D), where A is a substring of C and B is a substring of D, is also missing.

We are now ready to state the following theorem.

Theorem 1. *The missing patterns problem on a sequence of length n for patterns of the same length can be solved in $O(\alpha n \log\log_\sigma n)$ time and $O(n)$ space, where σ is the alphabet size.*

Proof. Recall that there exists a missing pattern pair (A, B), where $a = b = m \leq \lceil \log_\sigma n \rceil$. Therefore, such missing pair can be found in linear time by Proposition 1. In order to find the shortest pair, we can do binary search on the pattern length $1 \dots m-1$ and apply Algorithm 1 for each length. From the Monotonicity Property we are guaranteed to output the shortest missing pattern pair of the same length in $O(\alpha n \log\log_\sigma n)$ time and $O(n)$ space. □

3.3 Finding Missing Pairs

We now consider the problem when the two patterns do not necessarily have the same length. From Lemma 1, there exists a missing pattern pair (A, B), where $a = m$ and $b = 1$ for a combined pair length of $m + 1$. Recall that $m \leq \lceil \log_\sigma n \rceil$ is the length of the shortest missing pattern of the input sequence T and can be found in linear time and space. Such missing pattern can be combined with any non-empty pattern to form a missing pattern pair. For $\alpha \geq m$, it is easy to see that for any missing pattern pair (A, B) of length $\leq m$, the concatenation of A and B should also be missing, otherwise A and B occur at a distance $\leq \alpha$. Therefore, for any missing pattern pair (A, B), $a + b \geq m$.

Theorem 2. *The missing patterns problem on sequence of length n can be solved in $O(\alpha n \log_\sigma n)$ time and $O(n)$ space, where σ is the alphabet size.*

Proof. We showed that the shortest missing pattern pair is of length at least m and at most $m + 1$. To find if a pattern pair (A, B) of length m is missing it is enough to verify all possible combinations of $a + b = m$. This can be done by applying $m = O(\log_\sigma n)$ times Algorithm 1. Therefore, we obtain the required running time. □

The above analysis assumes $\alpha \geq m$. In the case when $\alpha < m$, there is also a solution to take in consideration of total length $2\alpha + 1$. Let $G, T \in \Sigma$ be two arbitrary letters of the input alphabet. Consider the pattern pair

$$(\underbrace{G \dots G}_{\alpha+1}, \underbrace{T \dots T}_{\alpha}) \ .$$

Trivially, it is a missing pair since the two patterns cannot occur α-close.

Remark 1. Let the alphabet size σ be a constant. Since there are $\sigma^m = O(n)$ pattern pairs of combined length $m = O(\log_\sigma n)$, one can adapt the bit-table algorithm from [1] to match the above running time and space requirements.

We now present an algorithm with running time independent of the threshold parameter α. The algorithm finds for each pair of patterns (A, B) of given length, the smallest α_{AB} s.t. the two patterns occur α_{AB}-close. Therefore, a pattern pair is missing iff $\alpha_{AB} > \alpha$. The algorithm also finds the smallest $\alpha_{\min}$ s.t. all pairs are $\alpha_{\min}$-close.

Algorithm 2. The algorithm takes advantage from the fact that there are not too many pattern pairs of total length m. More precisely, there are at most $\sigma^{\lceil \log_\sigma n \rceil} < \sigma n$ such pairs. Again, we present the algorithm for fixed lengths of the pattern pairs (A, B) and adapt similar notation to Algorithm 1. We further assume $a + b = m$. The steps of the algorithm are as follow:

1. Let L be an array of length N_1, where $L[h]$ is the list of occurrence positions in T of the pattern of length a mapped to the integer h. Let R be an array of length N_2, where $R[h']$ is the list of occurrence positions in T of the pattern of length b mapped to the integer h'.

2. For each pattern pair (A, B), merge efficiently the corresponding lists of occurrence positions (which are sorted by construction) to find the closest occurrence of A and B and therefore α_{AB}.

Analysis. The algorithm clearly requires $O(n)$ space, and we claim it takes $O((\sigma + \log n)n\sqrt{n})$ time. Step 1 of the algorithm can be performed in $O(n)$ time by scanning T from left to right. We now analyze Step 2. For a given pattern, we will call its list of occurrence positions *long* if it has length at least $\sqrt{n}$. We note that there are at most $\sqrt{n}$ long lists in L since the total length of all lists is at most n. Similarly, there are at most $\sqrt{n}$ long lists in R. All pairs of lists that are not long can be merged in $O(\sigma n\sqrt{n})$ time using merge sort since there are $O(\sigma n)$ such pairs. Let I be the set of indices of long lists in L, i.e. for all $h \in I$, $|L[h]| \geq \sqrt{n}$. Fix $h \in I$. The list $L[h]$ can be merged using binary search with all lists in R in time proportional to $\sum_{h'} |R[h']| \log |L[h]| = O(n \log |L[h]|)$. Summing over $h \in I$ we obtain $n \sum_{h \in I} \log |L[h]| = O(n\sqrt{n} \log n)$ since $|I| \leq \sqrt{n}$ and each list is of length at most n. Applying the same argument for the long lists in R we obtain the desired running time.

The next theorem follows by an argument analogous to the proof of Theorem 2 but applying Algorithm 2.

Theorem 3. *The missing pattern problem on sequence of length n can be solved in $O((\sigma + \log n)n\sqrt{n} \log_\sigma n)$ time and $O(n)$ space, where σ is the alphabet size.*

4 Extensions to the Missing Pattern Pair Problem

We discuss the following two extensions to the problem of finding missing pattern pairs of fixed lengths. First, we show how to find missing pairs when the patterns are restricted to occur at certain regions of the input sequence T. Next, in addition, we allow the patterns to be of length equal or greater than m. We describe the required changes to Algorithm 1, and then state how it generalizes to the problem of finding the shortest pattern pairs of the same or different lengths.

Localized Patterns. Let P_L (P_R) be the set of positions where pattern A (B) of pair (A, B) can occur. The sets can be specified as interval lists or bit-tables. For simplicity we assume the latter representation, which can be obtained from the interval lists in $O(n)$ time and space[3]. We are interested in finding pattern pairs that occur at the restricted positions at a distance at most α, but do not occur α-close anywhere else.

We modify Algorithm 1 as follows. We restrict occurrence positions for A patterns in lists in L only to those in P_L in a straightforward manner. In the

[3] The conversion can be done even when the intervals are overlapping by storing for each position the number of intervals starting and ending at that position, and then scanning the array from left to right.

same fashion, in Step 3, we count for each pattern A, the distinct patterns B that occur at distance at most α. If there is a pattern missing, we do an additional pass to look for α-close unmarked pattern that start in P_R. It is not hard to see that with the described modifications the time and space requirements of the algorithm do not change.

Long Patterns. Since patterns are restricted to occur in the input sequence T, there are at most n candidate patterns for each A and B irrespective to their length. We can maintain the same framework of Algorithm 1 given a suitable (hash) function mapping valid A and B patterns to integers 0 to $O(n)$ corresponding to lists L (Step 1) and H (Step 2). We obtain the desired properties by computing the suffix tree of T and using the node indices corresponding to the patterns of length a and b in a standard way (see for details [14]). Computing the suffix tree only requires additional $O(n \log \sigma)$ time and $O(n)$ space [14].

We are now ready to state the following theorem.

Theorem 4. *The generalized missing patterns problem on a sequence of length n can be solved in*

- $O(\alpha n \ell \log \sigma)$ *time when the patterns are of the same length;*
- $O(\alpha n \ell^2 \log \sigma)$ *time when the patterns are allowed to have different lengths,*

where σ is the alphabet size, and ℓ is the total length of the output pair. In both cases the space requirement is $O(n)$.

Proof. (Sketch) Note that because of the condition that patterns must occur α-close at specific positions, the Monotonicity Property does not hold. We therefore need to run the extended Algorithm 1 for all possible combinations of pattern lengths up to ℓ. □

5 Experiments

We have performed preliminary tests with the human genome[4] to complement the results reported in [1] for the baker's yeast (*Saccharomyces cerevisiae*) genome. We set the threshold parameter α to a realistic value 5000 and searched for the shortest missing pattern pairs where the patterns are of the same length k. We found 238 pattern pairs for $k = 8$. Interestingly, the shortest pattern pairs found for the baker's yeast genome, which is about 250 times smaller, were also of length 8 [1]. From the 238 pattern pairs, 20 pairs are missing from both the human and the baker's yeast genome. Table 3 summarizes these missing pairs and the shortest distance between the patterns (or their reverse complements) of each pair in the corresponding genomes. For reference, the shortest (single) missing patterns from the human genome are of length 11 and are listed in Table 4. This is also surprising since the human genome length is roughly equal to 4^{16}.

[4] Available at ftp://ftp.ensembl.org/pub/current_human/

Table 3. Unordered missing pattern pairs in both the human and baker's yeast genomes for $k = 8$. The reverse complements of the shown pattern pairs are also missing.

Missing Pairs	Yeast α_{AB}	Human α_{AB}
(AATCGACG, CGATCGGT)	5008	6458
(CCGATCGG, CCGTACGG)	5658	6839
(CGACCGTA, TACGGTCG)	13933	7585
(CGACCGTA, TCGCGTAC)	5494	5345
(CGAGTACG, GTCGATCG)	5903	8090
(CGATCGGA, GCGCGATA)	6432	6619

Table 4. (Single) missing patterns from the human genome of length 11. The reverse complements of the shown patters are also missing.

Missing Patterns		
ATTTCGTCGCG	CGGCCGTACGA	CGCGAACGTTA
CCGAATACGCG	CGTCGCTCGAA	CGTTACGACGA
CCGACGATCGA	CGACGCGATAG	GCGTCGAACGA
CGCGTCGATAG	CGATTCGGCGA	TATCGCGTCGA

An implementation in Java of the used software is available at the author's homepage[5]. The program needed about 3 hours to process the baker's yeast genome on a 1GHz machine, and about 30 hours for the human genome. The stop condition of step 3 of Algorithm 1, namely when all pattern pairs are discovered for the current pattern, provides a significant optimization in practice which allows the software to run only 10 times slower (rather than 250 times) for the human genome compared to the yeast genome.

6 Conclusions and Further Work

The *missing pattern discovery problem* was first introduced in [1] for optimal selection of adapter primers for nested PCR. In this paper, we presented more simple and efficient algorithms to solve the problem. The presented algorithms only require linear space and thus are efficiently implementable, whereas most algorithms in [1] take super-linear space. Our algorithms also have advantages for running time compared to those in [1], especially when the alphabet size σ is not constant. We implemented our algorithms and made experiments for the human genome and the baker's yeast genome, and we succeeded in finding shortest missing pairs of length 8 for both human and yeast genomes. In addition, we studied an extended version of the problem where patterns in the pair occur at certain positions at a distance at most α, but do not occur α-close anywhere else, in the input sequence.

[5] http://www.cis.upenn.edu/∼angelov

As a generalization of the missing pattern discovery problem, the following problem that allows mismatches is worth to consider: Given sequence T, distance α, and error parameter e, find pattern pair (A, B) such that any occurrence of A and B within e mismatches in T is not α-close. In [13], they presented some algorithms to discover structured motifs with errors in the Hamming distance metric. Since the algorithms of [1] and [13] are both based on suffix trees, it might be possible to solve the above general missing pattern discovery problem by combining these algorithms.

7 Acknowledgement

The authors would like to thank Teemu Kivioja of VTT Biotechnology and Veli Mäkinen of Bielefeld University, for their contributions to the pioneering work on finding missing pattern pairs in [1] and their fruitful comments to this work.

References

1. Inenaga, S., Kivioja, T., Mäkinen, V.: Finding missing patterns. In: Proc. 4th Workshop on Algorithms in Bioinformatics (WABI'04). Volume 3240 of LNCS., Springer-Verlag (2004) 463–474
2. Apostolico, A.: Pattern discovery and the algorithmics of surprise. In: Artificial Intelligence and Heuristic Methods for Bioinformatics. (2003) 111–127
3. Shinohara, A., Takeda, M., Arikawa, S., Hirao, M., Hoshino, H., Inenaga, S.: Finding best patterns practically. In: Progress in Discovery Science. Volume 2281 of LNAI., Springer-Verlag (2002) 307–317
4. Shimozono, S., Shinohara, A., Shinohara, T., Miyano, S., Kuhara, S., Arikawa, S.: Knowledge acquisition from amino acid sequences by machine learning system BONSAI. Transactions of Information Processing Society of Japan **35** (1994) 2009–2018
5. Bannai, H., Inenaga, S., Shinohara, A., Takeda, M., Miyano, S.: Efficiently finding regulatory elements using correlation with gene expression. Journal of Bioinformatics and Computational Biology **2** (2004) 273–288
6. Baeza-Yates, R.A.: Searching subsequences (note). Theoretical Computer Science **78** (1991) 363–376
7. Hirao, M., Hoshino, H., Shinohara, A., Takeda, M., Arikawa, S.: A practical algorithm to find the best subsequence patterns. In: Proc. 3rd International Conference on Discovery Science (DS'00). Volume 1967 of LNAI., Springer-Verlag (2000) 141–154
8. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovering frequent episode in sequences. In: Proc. 1st International Conference on Knowledge Discovery and Data Mining, AAAI Press (1995) 210–215
9. Hirao, M., Inenaga, S., Shinohara, A., Takeda, M., Arikawa, S.: A practical algorithm to find the best episode patterns. In: Proc. 4th International Conference on Discovery Science (DS'01). Volume 2226 of LNAI., Springer-Verlag (2001) 435–440
10. Inenaga, S., Bannai, H., Shinohara, A., Takeda, M., Arikawa, S.: Discovering best variable-length-don't-care patterns. In: Proc. 5th International Conference on Discovery Science (DS'02). Volume 2534 of LNCS., Springer-Verlag (2002) 86–97

11. Takeda, M., Inenaga, S., Bannai, H., Shinohara, A., Arikawa, S.: Discovering most classificatory patterns for very expressive pattern classes. In: Proc. 6th International Conference on Discovery Science (DS'03). Volume 2843 of LNCS., Springer-Verlag (2003) 486–493
12. Eskin, E., Pevzner, P.A.: Finding composite regulatory patterns in DNA sequences. Bioinformatics **18** (2002) S354–S363
13. Marsan, L., Sagot, M.F.: Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. J. Comput. Biol. **7** (2000) 345–360
14. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)
15. Carvalho, A.M., Freitas, A.T., Oliveira, A.L., Sagot, M.F.: A highly scalable algorithm for the extraction of cis-regulatory regions. In: Proc. 3rd Asia Pacific Bioinformatics Conference (APBC'05), Imperial College Press (2005) 273–282
16. Arimura, H., Arikawa, S., Shimozono, S.: Efficient discovery of optimal word-association patterns in large text databases. New Generation Computing **18** (2000) 49–60
17. Arimura, H., Asaka, H., Sakamoto, H., Arikawa, S.: Efficient discovery of proximity patterns with suffix arrays (extended abstract). In: Proc. the 12th Annual Symposium on Combinatorial Pattern Matching (CPM'01). Volume 2089 of LNCS., Springer-Verlag (2001) 152–156
18. Liu, X., Brutlag, D., Liu, J.: BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. In: Pac. Symp. Biocomput. (2001) 127–138
19. Bannai, H., Hyyrö, H., Shinohara, A., Takeda, M., Nakai, K., Miyano, S.: An $O(N^2)$ algorithm for discovering optimal Boolean pattern pairs. IEEE/ACM Transactions on Computational Biology and Bioinformatics **1** (2004) 159–170
20. Inenaga, S., Bannai, H., Hyyrö, H., Shinohara, A., Takeda, M., Nakai, K., Miyano, S.: Finding optimal pairs of cooperative and competing patterns with bounded distance. In: Proc. 7th International Conference on Discovery Science (DS'04). Volume 3245 of LNCS., Springer-Verlag (2004) 32–46
21. Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P.: Molecular Biology of the Cell, fourth edition. Garland Science (2002)
22. Karp, R., Rabin, M.: Efficient randomized pattern-matching algorithms. IBM Journal of Research and Development **31** (1987) 249–260