Efficient composite pattern finding from monad patterns

Jianjun Zhou, Jörg Sander and Guohui Lin*

Bioinformatics Research Group,
Department of Computing Science,
University of Alberta, Edmonton,
Alberta T6G 2E8, Canada
E-mail: jianjun@cs.ualberta.ca
E-mail: joerg@cs.ualberta.ca
E-mail: ghlin@cs.ualberta.ca
*Corresponding author

Abstract: Automatically identifying frequent composite patterns in DNA sequences is an important task in bioinformatics, especially when all the basic elements (or monad patterns) of a composite pattern are weak. In this paper, we compare one straightforward approach to assemble the monad patterns into composite patterns to two other rather complex approaches. Both our theoretical analysis and empirical results show that this overlooked straightforward method can be several orders of magnitude faster. Furthermore, different from the previous understandings, the empirical results show that the runtime superiority among the three approaches is closely related to the insignificance of the monad patterns.

Keywords: pattern finding; monad pattern; composite pattern; runtime complexity; bioinformatics.

Reference to this paper should be made as follows: Zhou, J., Sander, J. and Lin, G. (2007) 'Efficient composite pattern finding from monad patterns', *Int. J. Bioinformatics Research and Applications*, Vol. 3, No. 1, pp.86–99.

Biographical notes: Jianjun Zhou is a PhD student in Computing Science at the University of Alberta. He received his MSc Degree from the same university in 2003. His research interests include data mining and its applications in bioinformatics.

Jörg Sander received his PhD in Computer Science from the University of Munich in 1998. He joined the University of Alberta as an Assistant Professor of Computing Science in July 2001. His research interests are in the areas of knowledge discovery in databases, spatial and spatio-temporal databases, and bioinformatics. His current focus is in the sub-areas of Clustering, Spatial Data Mining, Data Mining in Biological Databases, and Spatio-Temporal Indexing and Querying.

Guohui Lin received his PhD in Theoretical Computer Science from the Chinese Academy of Sciences in 1998. He joined the University of Alberta as an Assistant Professor of Computing Science in July 2001. His research interests include bioinformatics, computational biology, and algorithm design and analysis, and the recent work focuses on algorithmic developments for protein structure determination and comparison, whole genome phylogenetic analysis, RNA structure prediction and comparison, and putative gene finding. He is a member of ACM and a member of IEEE Computer Society.

1 Introduction

Finding composite DNA patterns is an interesting topic in bioinformatics research and has received much attention recently (Marsan and Sagot, 2000; Eskin and Pevzner, 2002; Carvalho et al., 2005). Given a set of DNA sequences, a *composite* DNA pattern is a combination of two or more frequent patterns that co-occurs more than a given number of times in the set of sequences. Each of the individual frequent patterns is called a *monad* pattern (Eskin and Pevzner, 2002), which is required to appear frequent enough in the set of sequences. Finding composite patterns is computationally challenging (Eskin and Pevzner, 2002), typically when one or all of the involved monad patterns are *weak* (or *insignificant*), i.e., there are too many candidate monad patterns resulting in too many possible combinations.

One solution to the composite pattern finding problem is to find out all candidate monad patterns and then to assemble them into composite patterns by scanning a set of positions behind or in front of a monad pattern for co-occurring monad patterns. Such an approach was first mentioned in Marsan and Sagot (2000), which is also the pioneering work on the composite pattern finding problem. However, Marsan and Sagot (2000) rejected this straightforward method based on over-estimated theoretical upper bounds of its runtime and space complexity. Instead, they proceeded to study a suffix tree based algorithm for the composite pattern finding problem, for which they found better upper bounds of runtime and space complexity. The suffix tree based algorithm was further extended by Eskin and Pevzner (2002) and Carvalho et al. (2005), to a prefix tree based algorithm MITRA-Count and an improved algorithm RISO based on a special suffix tree, respectively.

In this paper, we study the above straightforward approach for composite pattern finding. We do this carefully to provide much tighter theoretical upper bounds on the runtime and space complexity. We will show later that, our analysed upper bound on the runtime turns out to be better than the suffix/prefix tree based algorithms, while the space complexity remains to be worse. We call our approach *ECOMP*, which stands for *Efficient COMPosite pattern finding*. We choose to compare the performance of ECOMP with MITRA-Dyad and RISO, which are the best implementations of the prefix and suffix tree based composite pattern finding algorithms, respectively. The experiments were done on both synthetic and real datasets with various parameter settings. The results confirmed one previous observation that ECOMP would be superior to MITRA-Dyad when the gaps between the monad patterns in the composite patterns are large, and also showed that ECOMP is superior even when the gaps are small but the involved monad patterns are weak. One general conclusion from our experimental results is that among these three algorithms, which one is superior to the others is closely related to the insignificancy of the involved monad patterns. Typically, when all involved monad patterns are insignificant, ECOMP can be up to several orders of magnitude faster than MITRA-Dyad and RISO.

For simplicity, in the remaining part of the paper we will only consider the composite patterns consisting of two monad patterns, i.e., *dyad patterns*. In the next section, we will give more detailed definitions related to composite patterns. In Section 3, we will describe the algorithm ECOMP in details. We will show in Section 4 that ECOMP has a better theoretical runtime than the one given in Marsan and Sagot (2000). We compare ECOMP with MITRA-Dyad and RISO on both synthetic and real datasets and report the results in Section 5. Section 6 concludes the paper with some further discussions.

2 Preliminaries

Genes having similar functions usually are controlled by common regulatory elements. Identifying such common regulatory elements can be formulated as finding frequent patterns for a given set of sequences. The study on such a pattern finding problem was started more than a decade ago and is still a hot topic in bioinformatics research. For the common regulatory element identification application, one starts with a set of genes that have similar functions to collect the upstream regions of all these genes, and then to search for frequent patterns that occur more than a given number of times in the collected sequences.

Such a pattern finding problem is both biologically and computationally interesting and challenging. On one hand, unlike frequent patterns in some other fields of studies such as frequent itemsets in association rule mining (Agrawal and Srikant, 1994), frequent DNA patterns usually contain mutations in their occurrences. In this sense, the target patterns are more like profiles and therefore identifying them could be computationally very expensive (Pevzner and Sze, 2000). On the other hand, fast DNA pattern finding algorithms are desired for high-throughput purpose such as a phase in microarray data analysis.

Most of the previous research in the literature has focused on finding so-called *monad* patterns (Eskin and Pevzner, 2002). Essentially, a monad pattern is a relatively short DNA string that appears (allowing a certain degree of mutations) in a given set of sequences more than a given number of times. With no intention to give a full survey here, we only name a few well known monad pattern finding algorithms: Gibbs sampling (Lawrence et al., 1993), MEME (Bailey and Elkan, 1995), CONSENSUS (Hertz and Stormo, 1999), WINNOWER (Pevzner and Sze, 2000), PROJECTION (Buhler and Tompa, 2002), and MULTIPROFILER (Keich and Pevzner, 2002). These monad pattern finding algorithms can be roughly divided into two categories based on the *models* assumed on the patterns, machine learning (or statistical) models and mismatch models. WINNOWER (Pevzner and Sze, 2000) and PROJECTION (Buhler and Tompa, 2002) are two monad pattern finding algorithms based on mismatch models. In this paper, we assume mismatch models too, which model each monad pattern as a contiguous string S and an expression of the form $(l, d) - k$, where l is the length of S, and S has at least k occurrences in the given sequences. An occurrence of pattern S is a length-l substring T (called an *l-mer*) in the given sequences that has at most d mismatches (mutations) with S. The requirement of "having at least k occurrences" has two different meanings, which may result in a slightly different way of counting the occurrences. In one meaning, pattern S must appear in at least k input sequences, regardless how many times it occurs in individual sequences; in the other meaning, pattern S must appear at least k times anywhere in the input sequences, that is, multiple occurrences in individual sequences are counted.

The monad pattern finding algorithms do not consider the co-occurrences of two or more patterns, which form a *composite* pattern. Composite patterns are biologically interesting, for example, they could form a group of transcription factors that collectively regulate the genes. However, detecting composite patterns is more challenging than finding monad patterns, as one or more of the involved monad patterns may be weak (or insignificant), i.e., hard to be distinguished from a vast number of candidate patterns existing in the input sequences.

Marsan and Sagot are probably the first to study the composite pattern finding problem with mismatch models (Marsan and Sagot, 2000). Their algorithm, SMILE, uses a suffix tree to extract patterns. Approaches that are based on machine learning models include CO-BIND (Thakurta and Stormo, 2001) using Gibbs sampling and an algorithm by van Helden et al. (2000) measuring the statistical significance of candidate monad pattern pairs. MITRA-Dyad (Eskin and Pevzner, 2002) and RISO (Carvalho et al., 2005) are based on mismatch models and they both further develop the idea in SMILE. SMILE and MITRA-Count use a suffix tree and a prefix tree to search the monad pattern space, respectively. To find composite patterns, MITRA-Dyad connects all possible DNA strings separated by gaps of a range of lengths, so that the problem becomes a monad pattern finding problem. RISO uses a special suffix tree called *factor tree* to explore the pattern space. It extends the connecting idea of MITRA-Dyad by introducing a new data structure called *box-links*, which is to connect the DNA strings in the factor tree. It should be noted that the general ideas underlying MITRA-Dyad and RISO on assembling monad patterns into composite patterns are the same. That is, they both extract one monad pattern first, and then focus on a window region of this monad pattern to scan for its sister monad patterns that co-occur more than a given number of times. Such a general approach avoids extracting monad patterns from the whole set of sequences, but it may need to repeat the monad pattern finding many times when all basic monad patterns are insignificant. We will compare ECOMP with MITRA-Dyad and RISO on both synthetic and real datasets.

3 The ECOMP algorithm

In this section, we formally describe the ECOMP algorithm for finding composite patterns of the form $(l_1, d_1) - [\text{dist}_{\min}, \text{dist}_{\max}] - (l_2, d_2) - k$ in a given set of N sequences each of length n. Such a form of composite patterns consists of two parts, the first part is a monad pattern of the form $(l_1, d_1) - k$ and the second part is a monad pattern of the form $(l_2, d_2) - k$, and these two parts are separated apart by at least $\text{dist}_{\min}$ nucleotides and at most $\text{dist}_{\max}$ nucleotides. The composite patterns must have at least k occurrences in the input sequences. During the presentation, we will point out the similarities and the differences between ECOMP and MITRA-Dyad.

3.1 ECOMP

The ECOMP algorithm consists of three steps of operations. In the first step, ECOMP extracts all candidate monad patterns in the target composite patterns, i.e., monad patterns of forms $(l_1, d_1) - k$ and $(l_2, d_2) - k$. To do this, ECOMP calls MITRA-Count (Eskin and Pevzner, 2002) for its efficiency. Essentially, to find $(l, d) - k$ monad patterns, MITRA-Count applies an exhaustive search in the whole pattern space using a prefix tree and prunes away those branches that do not have a minimal support of k. In more details, MITRA-Count starts from an empty root node and grows the prefix tree in a depth-first manner by appending a nucleotide to the current branch. Each branch forms a prefix. As long as there are at least k l-mers in the input sequences each has at most d mismatches to the prefix, MITRA-Count continues to extend it, or otherwise switches to the next branch in a depth-first manner. At the end, a prefix of length l in the tree is a pattern found. Experiments by Eskin and Pevzner (2002) show that this

prefix tree based monad pattern finding algorithm has better runtime than several existing monad pattern finding methods in the literature. In the second step, for each monad pattern of the form $(l_1, d_1) - k$, ECOMP scans the downstream window region $[\text{dist}_{\min}, \text{dist}_{\max}]$ of its every occurrence to count the occurrences of every other monad pattern of the form $(l_2, d_2) - k$. In the last step, ECOMP reports the found composite patterns, which are pairs of monad patterns having a count of occurrences greater than or equal to k.

In more details, ECOMP calls MITRA-Count in its first step to find all monad patterns of forms $(l_1, d_1) - k$ and $(l_2, d_2) - k$, each of which is accompanied with the starting positions of all its occurrences, ordered by increasing input sequence indices. In the second step, ECOMP uses the $(l_1, d_1) - k$ monad patterns as *query sources* to determine the $(l_2, d_2) - k$ monad patterns that are regarded as *query targets*. To do this, ECOMP builds an array of size Nn, of which the ith cell stores the list of target monad patterns having an occurrence starting at position i. Another array *stat*, whose size is equal to the number of target monad patterns, is also created. The jth cell in array *stat* is for the jth target monad pattern, and it has a field *count* that records the number of occurrences of the jth target monad pattern. ECOMP allocates a set *RelevantTargets* for collecting target monad patterns that co-occur with the source monad pattern under consideration. At each iteration, ECOMP examines one source monad pattern P by scanning the downstream window region $[\text{dist}_{\min}, \text{dist}_{\max}]$ associated with every occurrence $\overline{P}$ of P. Assuming $\overline{P}$ ends at position e, for every target monad pattern Q that has an occurrence starting at position i, where $e + \text{dist}_{\min} \le i \le e + \text{dist}_{\max}$, ECOMP performs the following operations depending on the meaning of 'k occurrences' (assuming Q is the jth target monad pattern):

- if the desired composite patterns are required to appear in at least k input sequences, then *stat*[j].*count* increases by only one per input sequence and Q is added to *RelevantTargets* at its first occurrence
- if the desired composite patterns are required to appear in at least k times anywhere in the input sequences, then *stat*[j].*count* increases by one at every occurrence and Q is added to *RelevantTargets* at its first occurrence.

After scanning for occurrences of P, if the jth target monad pattern Q is in *RelevantTargets* and *stat*[j].*count* $\ge k$, then a composite pattern composed of P and Q is found. ECOMP proceeds to re-initialise array *stat* and set *RelevantTargets*, and moves on to the next iteration to examine the next source monad pattern.

In the above description of ECOMP we assume that composite patterns are in the form $(l_1, d_1) - [\text{dist}_{\min}, \text{dist}_{\max}] - (l_2, d_2) - k$, that is, the $(l_2, d_2) - k$ monad pattern follows the $(l_1, d_1) - k$ monad pattern in the input sequences. When the physical order of these two parts is biologically irrelevant, e.g., composite regulatory elements are order independent and they function well as long as the elements are within a certain range of each other, either upstream or downstream, ECOMP can be easily adjusted to find them by simply adding another upstream query window. We remark that adding another query window increases the runtime of ECOMP only by a fraction, as all the target monad patterns have been identified. MITRA-Dyad and RISO have a different story. In order for MITRA-Dyad and RISO to find order-independent composite patterns, theoretically we may also add another upstream query window. However, since the runtime of MITRA-Dyad or RISO is very sensitive to (exponential in) the window size, adding a

new window is not really feasible for them – adding a new window is done by increasing the window size. The computational results presented in Section 5 confirm the above theoretical observations.

3.2 Stage-MITRA

MITRA-Count is designed for monad pattern finding and it is extended to MITRA-Dyad for dyad pattern finding. Essentially, MITRA-Dyad reduces the dyad pattern finding problem to a monad pattern finding, as detailed in the following. The target dyad patterns in MITRA-Dyad are in the form $(l_1 - [\text{dist}_{\min}, \text{dist}_{\max}] - l_2, d_1 + d_2) - k$, which are composite patterns consisting of two monad patterns of lengths l_1 and l_2, respectively, separated apart by at least $\text{dist}_{\min}$ nucleotides and at most $\text{dist}_{\max}$ nucleotides, allowing in total at most $d_1 + d_2$ mismatches, and occurring at least k times in the input sequences. Note that this form of composite patterns differs from those of the form $(l_1, d_1) - [\text{dist}_{\min}, \text{dist}_{\max}] - (l_2, d_2) - k$. In fact, the latter is a special case of the former. In this regard, MITRA-Dyad alters a bit the original composite pattern finding problem first studied in Marsan and Sagot (2000). To find $(l_1 - [\text{dist}_{\min}, \text{dist}_{\max}] - l_2, d_1 + d_2) - k$, patterns, MITRA-Dyad concatenates each l_1-mer with the l_2-mer that is downstream s nucleotides away, for every $s \in [\text{dist}_{\min}, \text{dist}_{\max}]$, to form an $(l_1 + l_2)$-mer. Then MITRA-Dyad proceeds to find monad patterns of the form $(l_1 + l_2, d_1 + d_2) - k$. The worst case runtime complexity of MITRA-Dyad grows exponentially in the number of allowed mismatches (Sagot, 1998; Eskin and Pevzner, 2002). Therefore, MITRA-Dyad spends an exponential amount of more time than ECOMP (i.e., $O(3^{d_1+d_2})$ (Eskin and Pevzner, 2002) vs. $O(3^{d_1} + 3^{d_2}))$ for finding the desired dyad patterns. Nonetheless, when the source monad patterns are significant, MITRA-Dyad works well even if the target monad patterns are weak.

The main difference of ECOMP compared to MITRA-Dyad is to find both parts in the composite patterns at the first step using independent parameters. Subsequently, ECOMP may choose the significant part as the source to determine its targets, downstream or upstream. We integrated this idea into MITRA to firstly find the source monad patterns using the model $(l_1, d_1) - k$, and then to proceed to find the target $(l_2, d_2) - k$ monad patterns in the downstream window regions of the occurrences of each source monad pattern. This hybrid method is called *Stage-MITRA*. Note that in the composite patterns found by Stage-MITRA, there are at most d_1 mismatches in the first part, and there are at most d_2 mismatches in the second part, i.e., they are $(l_1, d_1) - [\text{dist}_{\min}, \text{dist}_{\max}] - (l_2, d_2) - k$ patterns. Correspondingly, Stage-MITRA is expected to run faster than MITRA-Dyad. The experimental results presented in Section 5 confirmed our expectation by showing that Stage-MITRA is up to an order of magnitude faster than the original MITRA-Dyad.

The output composite patterns by ECOMP and Stage-MITRA satisfy stricter constraints than the output composite patterns found by MITRA-Dyad, in that the numbers of mismatches in the first and the second parts are at most d_1 and d_2, respectively. We remark that SMILE and RISO are also designed for finding composite patterns of the form $(l_1, d_1) - [\text{dist}_{\min}, \text{dist}_{\max}] - (l_2, d_2) - k$, which is more reasonable than the form used in MITRA-Dyad. In general, the model on the composite patterns in MITRA-Dyad is too loosely defined, and MITRA-Dyad might return too many false positives when the source monad patterns are extremely strong and the target monad

patterns are extremely weak. Between ECOMP and Stage-MITRA, ECOMP outperforms Stage-MITRA in general, since ECOMP only solves two independent monad pattern finding problems while Stage-MITRA might need to solve a huge number of them. Nevertheless, in some extreme situations where the target part of the composite pattern is extremely insignificant and the source part is extremely strong, Stage-MITRA could outperform ECOMP. The explanation is that because the source pattern is so strong, the space of target patterns reduces dramatically due to the window constraint, and Stage-MITRA may benefit from this fact more than ECOMP by performing only a few monad pattern findings in small search spaces.

4 Theoretical runtime analysis

In this section, we provide the theoretical worst case runtime complexity analysis for the different ways of assembling monad patterns in Stage-MITRA and ECOMP. We first list the notations used in the analysis:

n:	Length of input sequences
N:	Number of input sequences
$\mathcal{V}(l, d)$:	Maximum number of l-mers at a Hamming distance of at most d from another l-mer
q:	Window size, $q = \text{dist}_{\max} - \text{dist}_{\min} + 1$
n_x:	Number of $(l_1, d_1) - k$ patterns in the input sequences
I_{x_i} :	Number of occurrences of the ith $(l_1, d_1) - k$ pattern in the input sequences
A_x:	Average number of occurrences for an $(l_1, d_1) - k$ pattern in the input sequences
n_y:	Number of $(l_2, d_2) - k$ patterns in the input sequences
A_y:	Average number of occurrences for an $(l_2, d_2) - k$ pattern in the input sequences
C_{ECOMP}:	Worst case runtime complexity of ECOMP in assembling monad patterns
$C_{\text{Stage-MITRA}}$:	Worst case runtime complexity of Stage-MITRA in assembling monad patterns.

Theorem 4.1: *Given all $(l_1, d_1) - k$ and $(l_2, d_2) - k$ monad patterns and their occurrences in the input sequences. The runtime complexity of Stage-MITRA and ECOMP for assembling monad patterns into composite patterns satisfying the model $(l_1, d_1) - [dist_{min}, dist_{max}] - (l_2, d_2) - k$ are*

$$C_{\text{Stage-MITRA}} \in O(q \sum_i I_{x_i}^2 \mathcal{V}(l_2, d_2))$$

and

$$C_{\text{ECOMP}} \in O(q \sum_i I_{x_i} \mathcal{V}(l_2, d_2))$$

respectively.

Proof: Sagot showed that the worst case runtime complexity of MITRA-Count for finding $(l, d) - k$ monad patterns is $O(nN^2\mathcal{V}(l, d))$ (Sagot, 1998; Eskin and

Pevzner, 2002). Stage-MITRA first extracts all $(l_1, d_1) - k$ monad patterns. When finding the second part $(l_2, d_2) - k$ monad patterns, it reduces to a monad pattern finding problem in the window region of every occurrence of an $(l_1, d_1) - k$ monad pattern. For finding the ith $(l_1, d_1) - k$ monad pattern, its runtime is $O(qI_{x_i}^2 \mathcal{V}(l_2, d_2))$. Consequently, the overall complexity is

$$C_{\text{Stage-MITRA}} \in O(q \sum_i I_{x_i}^2 \mathcal{V}(l_2, d_2)).$$

In ECOMP, for every occurrence of an $(l_1, d_1) - k$ monad pattern x_i, its downstream window region is scanned. From the definition of $\mathcal{V}(l_2, d_2)$, we have

$$C_{\text{ECOMP}} \in O(q \sum_i I_{x_i} \mathcal{V}(l_2, d_2)).$$

From

$$\sum_i I_{x_i}^2 \geq \left(\sum_i I_{x_i}\right)^2 \Big/ n_x = A_x \sum_i I_{x_i},$$

we conclude that in the worst case C_{ECOMP} is about an order of A_x less than $C_{\text{Stage-MITRA}}$. □

Note that in Marsan and Sagot (2000) there is an estimated upper bound of $n^2 N^2 q \mathcal{V}(l_1, d_1) \mathcal{V}(l_2, d_2)$ on the runtime of a naive approach, which is similar to ECOMP in spirit. A careful look at it reveals that such an upper bound is over-estimated. If including the runtime for extracting monad patterns, the runtime complexity of Stage-MITRA and ECOMP are

$$O(q \sum_i I_{x_i}^2 \mathcal{V}(l_2, d_2) + nN^2 \mathcal{V}(l_1, d_1))$$

and

$$O(q \sum_i I_{x_i} \mathcal{V}(l_2, d_2) + nN^2 \mathcal{V}(l_1, d_1) + nN^2 \mathcal{V}(l_2, d_2))$$

respectively. When both parts in the composite patterns are not strong, then the runtimes of Stage-MITRA and ECOMP are both dominated by the time complexity for assembling the monads. In this case, Theorem 1 says that the runtime of ECOMP is linear in the window size q, the number of source monad patterns n_x, and the average number of occurrences of source monad patterns A_x. We remark that ECOMP trades memory for speed, as it can be seen that the space complexity of ECOMP is $\Theta(n_x A_x + n_y A_y + nN)$, while that of MITRA-Count is only $\Theta(nN)$. However, since ECOMP retrieves query patterns one by one in a sequential way and only does random access on the storage of the target patterns, we can always use the stronger monad patterns as target patterns, so that even if the weaker patterns can only be stored in secondary memory, the access of secondary memory is a sequential one which is well known in computer science to be much faster than random access of secondary memory. If both parts are too weak to be stored in memory (i.e., both $n_x A_x$ and $n_y A_y$ are extremely large), then MITRA-Dyad may not work as well since in its time complexity contains the factor of $n_x A_x^2$ $(q \sum_i I_{x_i}^2 \mathcal{V}(l_2, d_2) \geq q A_x \sum_i I_{x_i} \mathcal{V}(l_2, d_2) = q n_x A_x^2 \mathcal{V}(l_2, d_2))$.

5 Computational experiments and discussions

We have conducted a series of computational experiments on both simulated DNA data and real biological data to compare the performance of ECOMP with MITRA-Dyad (Eskin and Pevzner, 2002), RISO (Carvalho et al., 2005), and Stage-MITRA. As the experimental results in Carvalho et al. (2005) have shown that RISO improves over SMILE (Marsan and Sagot, 2000) by several orders of magnitude in runtime, we do not include SMILE in our comparison. Also, since MITRA-Dyad finds composite patterns in the form other than that in ECOMP, RISO, and Stage-MITRA, we will mainly compare ECOMP with Stage-MITRA and RISO to show the performance difference between the ways of assembling monad patterns into composite patterns. The computational results in the following show that ECOMP is up to two orders of magnitude faster than Stage-MITRA and RISO, and is up to three orders of magnitude faster than MITRA-Dyad.

The program RISO was downloaded from the homepage of its authors (Carvalho et al., 2005). Due to the unavailability of MITRA-Count source code, we implemented the algorithm ourselves, according to the exact specification provided in Eskin and Pevzner (2002). In more details, the implementation of MITRA-Count uses the prefix tree data structure only (Eskin and Pevzner, 2002). Note that Eskin and Pevzner also proposed a hybrid method combining MITRA and a graph based approach WINNOWER (Eskin and Pevzner, 2002; Pevzner and Sze, 2000), called *MITRA-Graph*. Roughly speaking, WINNOWER is used in MITRA-Graph as a heuristic to speedup the monad pattern finding and eventually speedup the composite pattern finding because MITRA needs to find long monad patterns representing the composite patterns. Since the current interest is to compare two different ways of assembling monad patterns into composite ones, and also because MITRA-Graph has a much more complex implementation than MITRA-Count and according to its authors MITRA-Graph is not consistently faster than MITRA-count, we compared to MITRA-Count only in our experiments. For fair comparisons, MITRA-Dyad, Stage-MITRA and ECOMP were all implemented in C++ using the LEDA (Mehlhorn and Näher, 1995) library. RISO was implemented in C by its authors (Carvalho et al., 2005). The reported runtimes include the time for extracting the monad patterns. All experiments were performed on a Pentium IV 2.6 GHz Linux workstation with 1 GB of RAM.

5.1 Results on simulated data

Eskin and Pevzner defined the *dyad challenge* problem based on simulated data (Eskin and Pevzner, 2002). In this problem, sequences on an alphabet of four letters, mimicking DNA sequences, are randomly generated. In our experiment, we set the number of sequences to $N = 20$ and the length of each sequence to $n = 600$. Thirteen out of these 20 sequences were implanted with two (14, 4) motifs at random positions with a fixed distance of 20 nucleotides between them. To apply the pattern finding algorithms, the window was set to [20, 20], that is, the window size was only 1. Table 1 collects the runtimes of MITRA-Dyad for finding (14 – [20, 20] – 14, 8) – 13 patterns, and Stage-MITRA and ECOMP for finding (14, 4) – [20, 20] – (14, 4) – 13 patterns. It can be seen that the runtimes of Stage-MITRA and ECOMP are close on the challenge problem, and they are much less than that of MITRA-Dyad, which searches a much larger space.

Table 1 Running times on the Dyad Challenge problem

	MITRA-Dyad	*RISO*	*Stage-MITRA*	*ECOMP*
Running time	>5 hours	853.68 secs	492.19 secs	482.55 secs

5.2 Results on real biological data

We obtained one real biological dataset from Thakurta and Stormo (2001). This dataset consists of 11 gene sequences regulated by two binding sites URS1 and UASH. These 11 genes were divided into three groups. Group 1 contains five genes in which the distance between URS1 and UASH ranges from 19 to 37. Group 2 also contains five genes in which the distance between URS1 and UASH ranges from 83 to 111. Group 3 contains only one gene, for which the distance between URS1 and UASH is 336, and the UASH site is downstream of URS1 instead of upstream for the other genes. In these 11 genes, UASH is observed to be much weaker than URS1. For example, in Group 1, UASH is a (7, 1) – 4 pattern while URS1 is a (10, 2) – 5 pattern, and Group 1 contains 1452 and 453 monad patterns in these two models, respectively.

5.2.1 Results on Group 1

We followed the same experimental setup as in Eskin and Pevzner (2002), except that the support threshold for (10, 2) monad patterns (including URS1) was set to 4 instead of 5. This made the problem more challenging, as the number of (10, 2) – 4 monad patterns is 5472 compared to only 453 (10, 2) – 5 monad patterns. The runtimes (in seconds) of the four algorithms MITRA-Dyad, RISO, Stage-MITRA, and ECOMP for finding all (7, 1) – [17, 42] – (10, 2) – 4 and (10, 2) – [17, 42] – (7, 1) – 4 dyad patterns are collected in Table 2, where the dyad patterns must occur in at least four genes. It can be seen that the speedup of ECOMP over MITRA-Dyad is three orders of magnitude, and that Stage-MITRA already outperforms MITRA-Dyad two orders of magnitude.

When the dyad patterns are required to appear at least four times anywhere in the sequences, the runtimes increased a bit and they are summarised in Table 3. Note that in this case, the models became weaker. Table 3 shows again that ECOMP is orders of magnitude faster than MITRA-Dyad and Stage-MITRA.

Table 2 Running times in seconds of MITRA-Dyad, RISO, Stage-MITRA and ECOMP, on Group 1 genes, where the dyad patterns must occur in at least four genes. RISO does not search backward and so only one value is reported

Model	*MITRA-Dyad*	*RISO*	*Stage-MITRA*	*ECOMP*
(7, 1) – [17, 42] – (10, 2) – 4	258.21	45.39	16.46	0.44
(10, 2) – [17, 42] – (7, 1) – 4	197.04	N/A	14.09	0.44

Table 3 Running times in seconds of MITRA-Dyad, RISO, Stage-MITRA, and ECOMP, on Group 1 genes, where the dyad patterns must occur at least four times anywhere in the sequences. The runtime of RISO is not reported since it does not accept such kind of parameters

Model	*MITRA-Dyad*	*Stage-MITRA*	*ECOMP*
(7, 1) – [17, 42] – (10, 2) – 4	383.22	82.08	1.74
(10, 2) – [17, 42] – (7, 1) – 4	392.71	120.21	1.7

Tables 2 and 3 also show the phenomenon that the physical order of the monad patterns in the composite patterns may affect the runtimes of the algorithms. For MITRA-Dyad and Stage-MITRA, the difference can be observed by the non-symmetric runtime complexities of the two directions (see also Section 2). For ECOMP, the difference is insignificant in these cases. However, in extreme cases, the performance of ECOMP might be affected by the physical orders, due to the different amounts of memory required during the computation. In the extreme case where one monad pattern is very weak while the other is very strong, ECOMP could terminate in a short amount of time if the proper order is picked, while it might run out of memory using the other order. In one experiment, we have tested ECOMP by running it to find $(10, 2) – [17, 42] – (11, 4) – 5$ and $(11, 4) – [17, 42] – (10, 2) – 5$ models in Group 1 genes. It is known that the $(11, 4) – 5$ model is extremely weak (the number of $(11, 4) – 5$ monad patterns in Group 1 genes is 2,004,913, while that of $(10, 2) – 5$ is only 453). To find $(11, 4) – [17, 42] – (10, 2) – 5$ composite patterns ECOMP only needs 18 seconds. But when trying to find $(10, 2) – [17, 42] – (11, 4) – 5$ composite patterns, ECOMP did not terminate in one hour. The reason is that in this extreme case the weaker monad patterns can not be stored in main memory so that if we use the stronger monad patterns to search the weaker patterns, we end up with performing random access of secondary memory, as discussed in the previous section. We conclude that using the weaker monad patterns as the source patterns in ECOMP, a dramatically better runtime can be obtained.

In the other experiments to be reported next, MITRA-Dyad and Stage-MITRA were run in both directions and the better runtimes were reported. This is done under the consideration that in practice we would not know which of the two monad patterns of composite patterns is the stronger one before actually generating them, and therefore it is in favour of MITRA-Dyad and Stage-MITRA. For ECOMP, since we generate the monad patterns independently, we can decide which one is stronger and which one is weaker and we always use the weaker monad patterns as the query sources. For RISO, we are only able to run it in one direction.

5.2.2 Theoretical running time validation

We have designed experiments to validate the theoretical runtime analysis in Theorem 1. We have tested the window size ranging from small values 1–6 and then three larger values 11, 16, 21. The runtimes of the three algorithms RISO, Stage-MITRA, and ECOMP are plotted in Figure 1, where we can see that ECOMP outperforms Stage-MITRA and RISO in all cases. The plot also shows that the runtimes of Stage-MITRA and ECOMP grow linearly in the window size, consistent with the theoretical analysis.

Experiments have also been set up to validate the runtime of Stage-MITRA and ECOMP with respect to the change of n_x, the number of source monad patterns. In these experiments, the window size was fixed at 26 and $(10, 2) – 4$ was the source monad pattern model. There were in total 5472 source monad patterns found, and the first n_x of them were used to grow or to query the target monad patterns of the composite patterns. The runtimes of Stage-MITRA and ECOMP vs. n_x are plotted in Figure 2. The runtimes agree again with the theoretical analysis, showing that both runtimes grow linearly in n_x.

Figure 1 Runtimes of RISO, Stage-MITRA, and ECOMP vs. the window size

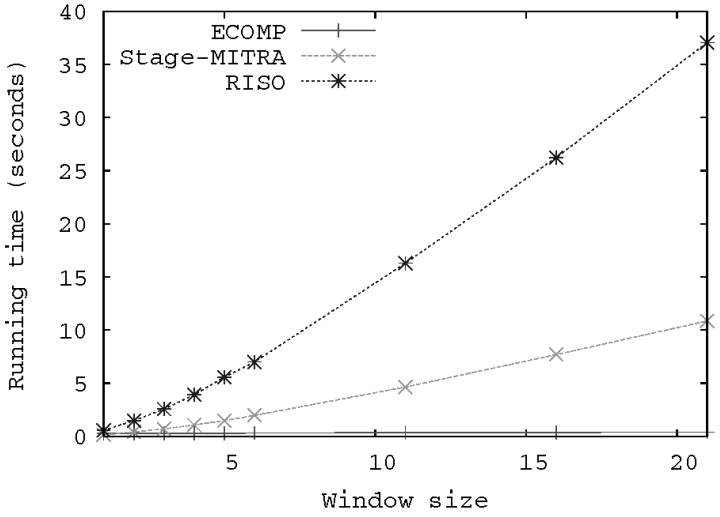
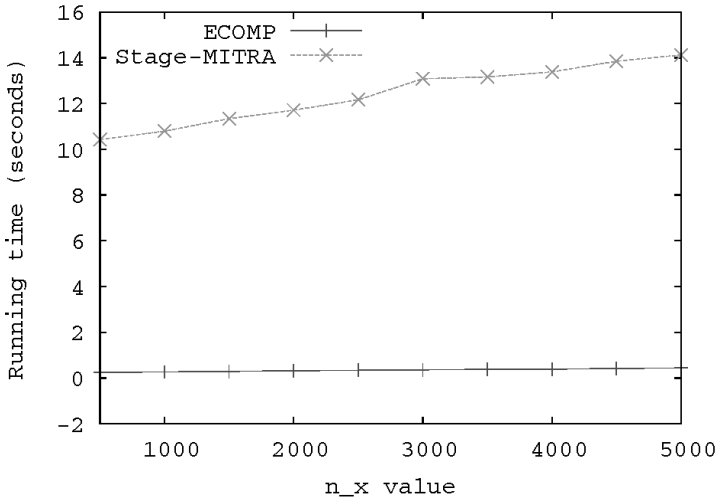
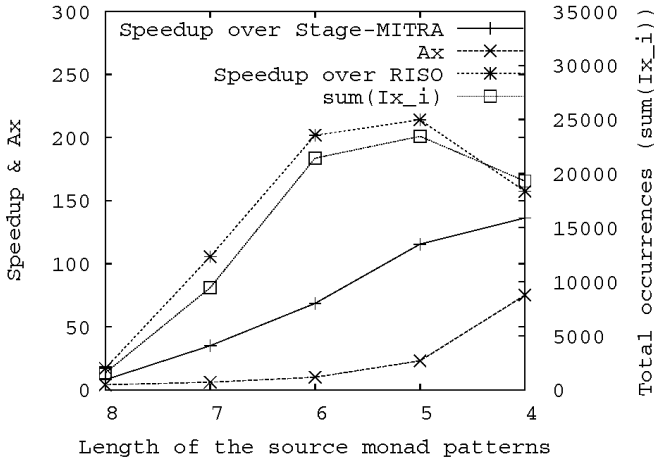


Figure 2 Runtimes of Stage-MITRA and ECOMP vs. the number of source monad patterns n_x



In the third set of experiments, we tested how RISO, Stage-MITRA, and ECOMP respond when the average number of occurrences of the source monad patterns, A_x, changes. We used $(l_1, 1) - [17, 42] - (10, 2) - 4$ as the composite pattern model, and tested l_1 from 8 to 4. Correspondingly, A_x increased. The growth of A_x with respect to l_1 and the speedup of ECOMP over Stage-MITRA and RISO with respect to l_1 are plotted in Figure 3, where we can see that the speedup of ECOMP over Stage-MITRA is larger than the growth of A_x. The trend of the speedup of ECOMP over RISO is similar to the trend of $\sum_i I_{x_i} = n_x \times A_x$, which indicates that the runtime of RISO is also closely related to n_x, the number of possible first monads. This confirms the theoretical observation that when all basic monad patterns are weak, ECOMP can be orders of magnitude faster than MITRA-Dyad, RISO, and Stage-MITRA.

Figure 3 Speedup of ECOMP over Stage-MITRA and RISO, A_x, the total number of occurrences of the source monad patterns $\sum_i I_{x_i}$, vs. the length of source monad patterns l_1



5.2.3 *Results on Groups 1 and 3*

The sequence in Group 3 is different from those in Group 1 in that the distance between URS1 and UASH is much larger and the physical order of the two sites is reversed. It is known that the URS1 and UASH patterns in the gene in Group 3 are more similar to those occurring in genes in Group 1, than to those occurring in genes in Group 2. In fact, if Groups 1 and 3 are merged together, then the composite pattern in all six genes can be modelled as (10, 2) – [17, 352] – (8, 2) – 6, disregarding the physical order of the two monad patterns. In this experiment, we have to use window size 336. We applied the order-independent version of ECOMP and Stage-MITRA to find the composite pattern. The runtime of ECOMP is 726.95 seconds, while Stage-MITRA ran out of CPU resource (>24 hours) on the same problem.

6 Conclusions

In this paper, we compared two different ways of assembling monad patterns into composite patterns. We have provided a better theoretical analysis on the runtime of an overlooked straightforward approach ECOMP, and we designed experiments to compare its performance with several complex composite pattern finding algorithms including MITRA-Dyad and RISO. The experimental results on synthetic and real-life data showed that ECOMP was up to two orders of magnitude faster than MITRA-Dyad and RISO, though theoretically MITRA-Dyad and RISO could perform better when one of the monad patterns in the composite patterns is extremely strong.

Besides, ECOMP also has the advantage that it can be used to extend any other monad pattern finding algorithm (assuming either the mismatch model or the machine learning model). We believe that the speed of ECOMP makes it a suitable choice for integration into high-throughput data analysis that requires fast composite pattern finding, such as to detect common transcription factors for sets of potentially co-regulated genes obtained by microarray cluster analysis.

Acknowledgements

The authors would like to thank J. Buhler for providing a software package for generating the simulated data. This work is supported partially by NSERC, CFI, and the University of Alberta.

References

Agrawal, R. and Srikant, R. (1994) 'Fast algorithms for mining association rules in large databases', *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 1994)*, September 12–15, Santiago de Chile, Chile, pp.487–499.

Bailey, T.L. and Elkan, C. (1995) 'Unsupervised learning of multiple motifs in biopolymers using expectation maximization', *Machine Learning*, Vol. 21, Nos. 1–2, pp.51–80.

Buhler, J. and Tompa, M. (2002) 'Finding motifs using random projections', *Journal of Computational Biology*, Vol. 9, No. 2, pp.225–242.

Carvalho, A.M., Freitas, A.T., Oliveira, A.L. and Sagot, M-F. (2005) 'A highly scalable algorithm for the extraction of CIS-regulatory regions', *Proceedings of the 3rd Asia-Pacific Bioinformatics Conference (APBC 2005)*, January 17–21, Singapore, pp.273–282.

Eskin, E. and Pevzner, P.A. (2002) 'Finding composite regulatory patterns in DNA sequences', *Proceedings of the 10th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB 2002)*, Bioinformatics, August 3–7, Edmonton, Alberta, Canada, Vol. 18, pp.S354–S363.

Hertz, G.Z. and Stormo, G.D. (1999) 'Identifying DNA and protein patterns with statistically significant alignments of multiple sequences', *Bioinformatics*, Vol. 15, No. 7, pp.563–577.

Keich, U. and Pevzner, P.A. (2002) 'Finding motifs in the twilight zone', *Proceedings of the 6th International Annual Conference on Research in Computational Molecular Biology (RECOMB 2002)*, April 18–21, Washington DC, USA, pp.195–204.

Lawrence, C.E., Altschul, S.F., Bogurski, M.S., Liu, J.S., Neuwald, A.F. and Wootton, J.C. (1993) 'Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment', *Science*, Vol. 262, pp.208–214.

Marsan, L. and Sagot, M-F. (2000) 'Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification', *Journal of Computational Biology*, Vol. 7, Nos. 3–4, pp.345–362.

Mehlhorn, K. and Näher, S. (1995) 'LEDA: a platform for combinatorial and geometric computing', *Communications of the ACM*, Vol. 38, No. 1, pp.96–102.

Pevzner, P.A. and Sze, S-H. (2000) 'Combinatorial approaches to finding subtle signals in DNA sequences', *Proceedings of the 8th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB 2000)*, August 19–23, San Diago, CA, USA, pp.269–278.

Sagot, M-F. (1998) 'Spelling approximate repeated or common motifs using a suffix tree', *Proceedings of the 3rd Latin American Theoretical Informatics Symposium (LATIN'98)*, LNCS 1776, April 20–24, Campinas, Brazil, pp.374–390.

Thakurta, D.G. and Stormo, G.D. (2001) 'Identifying target sites for cooperatively binding factors', *Bioinformatics*, Vol. 17, No. 7, pp.608–621.

van Helden, J., Rios, A.F. and Collado-Vides, J. (2000) 'Discovering regulatory elements in noncoding sequences by analysis of spaced dyads', *Nucleic Acids Research*, Vol. 28, pp.1808–1818.