

# We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900

Open access books available

185,000

International authors and editors

200M

Downloads

Our authors are among the

154

Countries delivered to

TOP 1%

most cited scientists

12.2%

Contributors from top 500 universities



WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index  
in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?  
Contact [book.department@intechopen.com](mailto:book.department@intechopen.com)

Numbers displayed above are based on latest data collected.  
For more information visit [www.intechopen.com](http://www.intechopen.com)



# Research on Pattern Matching with Wildcards and Length Constraints: Methods and Completeness

Haiping Wang, Taining Xiang and Xuegang Hu

Additional information is available at the end of the chapter

<http://dx.doi.org/10.5772/48574>

## 1. Introduction

The practical importance of the string matching problem should be obvious to everyone. For typical word-processing applications, immense amounts of work have been done on this subject. However, with the developments in bioinformatics (Cole et al., 2005), information retrieval (Califf et al., 2003), pattern mining (Xie et al., 2010; Ji et al., 2007; He et al., 2007), etc, sequential Pattern Matching with Wildcards and Length constraints (PMWL) has attracted more and more attention. It is not difficult to think up realistic cases where PMWL plays an important role. In Dan Gusfield's book (Gusfield, 1997), they give an example about *transcription factor* to illustrate the concept of wildcard. A *transcription factor* is a protein that binds to specific locations in DNA and regulates the transcription of the DNA into RNA. In this way, production of the protein that the DNA codes for is regulated. Many transcription factors are found and can be separated into families characterized by specific substrings containing wildcards. They use *Zinc Finger*, a common transcription factor as an example. It has the following signature:

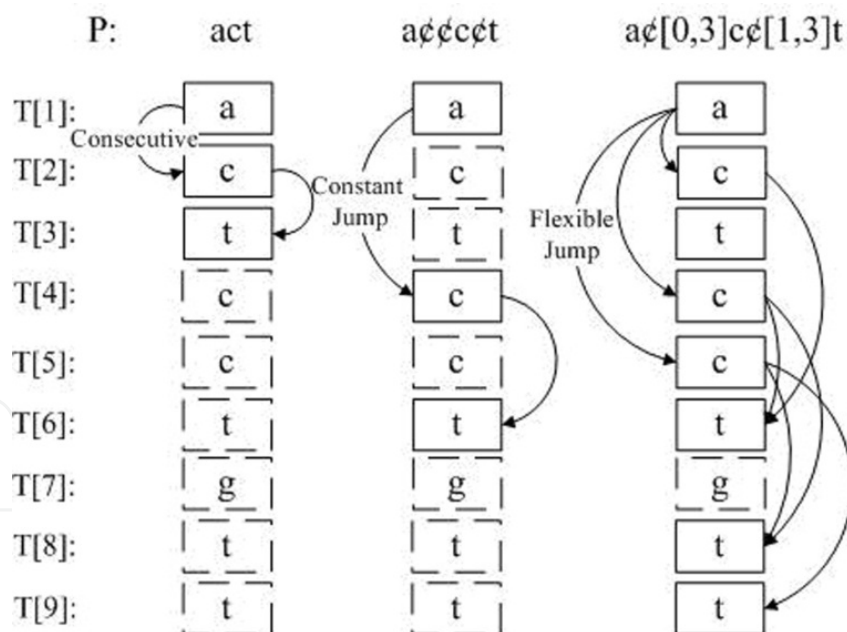
CYS $\epsilon$ CYS $\epsilon\epsilon\epsilon\epsilon\epsilon\epsilon\epsilon\epsilon\epsilon\epsilon\epsilon\epsilon\epsilon$ HIS $\epsilon$ HIS

Where CYS is the amino acid cysteine and HIS is the amino acid histidine. They also give a conclusion that if the number of wildcards is bounded by a fixed constant, the problem can be solved in linear time.

Another respective example is about *promoter*. In bioinformatics, *promoter* will help researchers to quickly locate the starting position of the intron from hundreds of millions of the sequence of ACGT. Among these promoters, TATA box is a common one (Manber & Baeza-Yates, 1991). It has very loose sequence specificity, so many TATA sequences are not

TATA box. As a result, indirect positioning by pairs of sites is needed. The commonly used one is *CAATCT* sequence. The DNA sequence *TATA* is a common promoter that often occurs after the sequence *CAATCT* within 30-50 wildcards. Therefore, matching patterns with wildcards becomes especially crucial in exploring valuable information from DNA sequences.

There are many applications that involve pattern matching with wildcards and various researches have provided many solutions to different forms of this problem. Fischer and Paterson were the first to generalize pattern matching with wildcards (Fischer & Paterson, 1974): given a pattern  $P$  and a text  $T$ , either of which may contain wildcards, denoted by  $\phi$ , the goal is to locate all  $P$ 's occurrences in  $T$ .  $\phi$  can match any letter in a given alphabet, such as  $a\phi\phi\phi t$ . Unlike previous work, Chen, et al. proposed a PMWL problem integrating two problems (Chen et al., 2006): one is complex local constraints which means the user can specify a different range of wildcards between each two consecutive letters of  $P$ , for example,  $a\phi[0,3]c\phi[1,3]t$ . Another one is global length constraints. The user can constrain the length of each matching substring of  $T$  in which  $P$  occurs. Therefore, flexible constraints of wildcards conduct flexible jump of the matching positions. The definition of PMWL problem is an extension of Fischer and Paterson's definition and the introduction of complex local constraints increase the flexibility. On one hand, this definition of pattern is more suitable for areas such as bioinformatics; on the other hand, the size of the matching candidate positions is in the exponential increment which greatly increases the complexity of the problem solving.



**Figure 1.** The flexibility and complexity of PMWL problem

From a view of practical point, they also proposed two issues: with and without the *one-off* condition (Chen et al., 2006; Min et al., 2009). In their problem definition, users have more flexibility to search on sequences and the *one-off* condition has both theoretical and practical significance. *One-off* condition means that every letter in  $T$  can be used once at most. In practical applications, with and without the *one-off* condition has practical meaning in specific

areas. For example, in sequential pattern analysis in data mining,  $P$  can be treated as a candidate shopping pattern, the user is interested in how frequently  $P$  occurs in one document, it makes sense to count each occurrence for once, what is more, the *one-off* condition also makes the problem solving possible. However, under the *one-off* condition, how to allocate limited text resource to each matching occurrences, in order to obtaining the maximum number of occurrences, belongs to optimization problem. In the allocation of resources, the matching of different letters in the pattern possess a strong correlation, which conducts the selection of matching positions in the combination of explosive growth. Since it is difficult to develop a complete matching strategy in this problem, almost existed algorithms for PMWL are using greedy matching strategies, which is the root reason why matching algorithm is not complete. This article will focus on SAIL algorithm (Chen et al., 2006) which is a representative algorithm for PMWL problem and will also describe RSAIL (Wang et al., 2010), SBO (Wu et al., 2011), BPBM (Guo et al., 2011) algorithm which are all designed to solve PMWL problem in different conditions. The each of above algorithms has its own characteristics in the data structures and matching strategies, which will be analyzed in this paper.

What is more, since the theoretic and practical importance of the definition of PMWL, we need to research the nature of this problem. To our best knowledge, there are still no efficient methods on this problem, because as for completeness of the problem, we still do not know whether it could be solved in polynomial time. In this article, we will research the completeness of PMWL under certain condition. In the traditional matching problem, description of pattern and text information is the key to the algorithm design, however, flexibility and complexity of the PMWL problem all depends on the pattern features, so this article will focus on pattern information, especially the pattern features including the size of alphabet, the length of pattern, the *gap* of wildcards in the pattern, etc. We will also investigate the relationship between pattern features and completeness, and use the approximate ratio judgment. Further more, since the definition itself is produced in realistic background, we need to consider the situation in real biological background and improve the solution of the problem. Based on the above, we choose this topic as a research object in this book.

This capture is organized as follows: In section 2 we will give the development, definition and application of PMWL problem; Section 3 will show the representative algorithms, we will introduce their structure, strategy, complexity and completeness; Section 4 will analyze the PMWL problem completeness based on pattern features. We will give our conclusions in section 5.

## 2. Pattern matching with wildcards and length constraints

The sequential pattern matching problem is to given a Text  $T$  and a pattern  $P$  as input, and output all the occurrences of  $P$  in  $T$ . After Fischer and Paterson's work, there are a variety of non-standard definitions of the pattern matching problem: the approximate matching (He et al., 2007), the swapped matching (Amir et al., 2000), the Parameterized matching (Amir et al., 2009), etc. They all belong to *Non-standard Stringology* problem (Muthukrishnan, 1994). Many of them are still open problems.

## 2.1. The development of PMWL problem

After years of development, these *Non-standard Stringology* problems always focus on a problem: that is, how to conduct the traditional pattern matching definition to be more flexible to adapt the development of application. The *don't cares* problem always focus on how to combine the wildcards and the pattern. After Fischer and Paterson's work, Cole et al. considered a slightly different problem (Cole et al., 2004), where instead of fixing the number of  $\phi$ s between two consecutive letters in  $P$  and  $T$ , they fixed the total number of  $\phi$ s in  $P$ . The disadvantage of these problem definitions is that the number of  $\phi$ s is a constant but not a range. This limits flexibilities for the user's queries. To alleviate the problem of a fixed number of  $\phi$ s, Kucherov et al. (Kucherov et al., 1995) proposed a solution to allow an unbounded number of  $\phi$ s between two consecutive letters in a given pattern. Given a set of such patterns, their objective is to find whether any of these patterns matches some substring of the text that does not contain any  $\phi$ . Obviously, allowing an unbounded number of  $\phi$ s still does not offer the users enough flexibilities to control their queries. Manber et al. (Manber & Baeza-Yates, 1991) proposed an algorithm for string matching with a sequence of wildcards. They considered the following problem: given two pattern strings  $P$  and  $Q$ , each of which consists of letters, and an integer  $g$ , all occurrences of the form  $P\phi_{0-g}Q$  in the text are returned. The number of  $\phi$ s between  $P$  and  $Q$  is in the range of  $[0, g]$ , and the text does not contain any  $\phi$ . This problem was so-called exact string matching with variable-length *don't cares*. Chen et al. sum up all these definitions into three conditions (Chen et al., 2006): firstly, there is a wildcard between two consecutive letters in  $P$ , for example  $A\phi[0,1]T\phi[0,2]G\phi[1,3]C$ ; secondly, every letter in  $T$  can only be used once for matching; thirdly, there is a global constraint to limit the matching occurrence length. We call the problem satisfying above definition PMWL problem, which has been used in approximate matching, pattern mining, information retrieval, etc.

## 2.2. The potential applications of the PMWL problem

1. Text Indexing: There is a large amount of hypertext information on the Internet. How to effectively obtain information that meet users' needs is becoming more and more urgent. Text indexing is a method to solve this problem. How to determine the position of user-specified pattern (may contain wildcards) is a challenge task.
2. Data stream is becoming more and more crucial in many new database applications such as data warehouse and sensor network. Mining dependence or association in large amounts of data flow has practical value and during which the sequential pattern matching with wildcards is the first and the most important step. In addition, in data mining, sequential pattern mining also search frequent patterns as in transaction sequence, a typical instance is the similar consuming pattern of many consumers, for example, buying a desktop, a laser printer, a digital camera and an LCD screen monitor in turn, between each of them exists a certain time interval. Mining such typical user mode, which is obviously a pattern matching with wildcards, will has a great influence on the market.

3. Network Security: Pattern matching methods in network security and intrusion detection need high performance. A complete IDS (Intrusion Detection System) based on Snort rules needs to optimize hundreds of rules and many of them need to do pattern matching efficiently for the entire data partition of a package. Efficient pattern matching and mining with wildcards constraints give the system administrator a more flexible and accurate solution to locate the suspicious users.

### 2.3. Problem statement for PMWL

**Definition 1** Let  $\Sigma$  be an alphabet,  $T = t_0t_1\dots t_{n-1} \in \Sigma^*$  is called a **text** of  $\Sigma$  where  $n = |T|$  is the length of  $T$ . A **pattern** is a tuple  $P = (p, g)$  where  $p = p_0p_1\dots p_{m-1}$  is a sequence of characters, which belong to the alphabet  $\Sigma$ , and  $g = g_0g_1\dots g_{m-2}$  is a sequence of wildcards. And  $m = |P|$  is the length of  $P$ . The interval of wildcards between  $p_i$  and  $p_{i+1}$  is denoted by  $g_i = g(N_i, M_i)$  where  $0 \leq i \leq m - 1$ , called the **local constraints**.  $N_i$  and  $M_i$  is the upper and lower limit of wildcard. Such as  $P = a\phi[1,3]g$ , where 1, 3 is respectively the lower and upper limit of local constraints.  $\phi[1,3]$  means the wildcards between  $a$  and  $g$  is referring to a string which length is 1~3. Given interval  $[minLen, maxLen]$ , set  $globalLength = t[a_{m-1}] - t[a_0] + 1$ , if  $globalLength \in [minLen, maxLen]$ , then it is called **global constraint** (Chen et al., 2006).

**Definition 2** Given a pattern  $P = (p, g)$ ,  $p = p_0p_1\dots p_{m-1}$ ,  $g_i = g(N_i, M_i)$ . The  $\max \{M_i - N_i\}$  where  $0 \leq i \leq m - 1$  is called the *gap* of local constraints, named **Gap** for short. For example,  $P = a\phi[0,2]g\phi[1,4]g$ , then  $gap = \max\{2 - 0, 4 - 1\} = 3$ .

PMWL problem can be defined by the above definition:

**Definition 3 PMWL problem** (Pattern Matching with Wildcards and Length constraints)

Pattern Matching with Wildcards and Length constraints meets the following conditions:

1.  $\phi$ s can occur between each two consecutive letters in pattern and are independent to each other;
2.  $\phi$ s between two consecutive letters can match a string which length is limited by *local constraints*, and the total length of pattern is limited by *global constraint*;
3. *One-off* condition is taken into consideration that every letter in  $T$  can only be used once for matching  $p_j$  ( $0 \leq j \leq m - 1$ ) and as soon as there exists one occurrence of  $P$  in  $T$  when  $T$  is being scanned from left to right it will be returned.

**Definition 4** Given a text  $T$  and a pattern  $P$ , if there is a sequence of matching positions  $A = (a_1, a_2, \dots, a_{m-1})$ , where  $t[a_i] = p[i]$  for every  $0 \leq i < m - 1$ , we say **A** is a matching **occurrence** of  $P$ . A set of occurrences  $A_1, A_2, \dots, A_t$  constitute an occurrence set  $U$  where  $t$  is the number of occurrences, and also named **matching number** in our paper.

**Definition 5** Let  $t$  be the matching number of  $A$ , if there is no occurrence set  $A'$  with the matching number  $t'$ , and  $t' > t$ , then  $A$  is called a **complete occurrence set**. If there is another occurrence set  $U$ , with the matching number  $t_u = t$ , the  $U$  is equivalent to  $A$ . Specially, if  $A$  is complete, and so is  $U$ . So the complete occurrence set is not always unique.



3. Algorithms for PMWL

The results of traditional matching algorithm are complete, so the focus of research is to improve the matching efficiency. As a kind of searching problem, the key to solving matching problem is how to use and extract information getting from text and pattern. KMP, BM algorithm uses automata to describe the pattern characteristics, and deposit information obtained from scanning during matching process into automata. Algorithm visits the automata, when the jump distance needs to be calculated, thus to avoid obtaining the pattern information repeatedly and to ensure the jump in matching process does not affect the final result. The basic idea the suffix tree is to use the tree structure to describe the text information, and to avoid scanning the same text repeatedly when matching a set of patterns. We believe that data structure and search strategy are crucial for traditional algorithms to access to information of text and pattern. Reasonable data structure is better to explore the potential of the computer, such as bit parallel technology, and can also be a more reasonable representation of the sequence information, such as automata. In addition, there exist the sliding window, indexes and other data structures. Reasonable matching strategy makes better use of sequence information. These strategies can approximately be divided into prefix searching, suffix searching and factor searching (Navarro & Raffinot, 2001).

	Characteristics	Representative algorithms	Methods	Remarks
prefix searching	Forward search to find the longest common prefix of text and pattern strings in searching window	KMP Shift-And	Deterministic automata Bit parallel, non-deterministic automata	Most of them are sliding window technique, the scope of algorithm application depends on the alphabet size and the pattern length
suffix searching	Backward search to find the longest common suffix of text and pattern strings, can skip some text characters, the difficulty is how to safely move the window	BM Horspool	Pre-calculation of the three functions used to determine the safe jumping distance Improve the function of the BM, can have a greater jump distance, especially suitable for larger alphabet	
factor searching	Backward search to determine whether the suffix of text in searching window is a substring of pattern string	BDM BNDM BOM	Suffix automaton Bit parallel Factor Oracle automaton, suitable for longer pattern	

Table 1. Analysis of the traditional pattern matching algorithms

As different extension of traditional matching problem, PMWL problem, approximate matching, and swap matching all belong to the *Non-standard Stringology* problem. Problems in this field mostly belong to the optimization problem, and most of them have not yet been completely solved, such as PMWL and approximate matching problems with wildcards etc. What PMWL and traditional matching problem have in common are:

1. From the view of algorithm itself, the data structures and matching strategies are as the key of algorithm design.
2. From the view of describing the object, how to effectively describe the patterns and text information is the key to solve the problem.

What PMWL and traditional matching problem have in difference are:

1. For PMWL, there is no complete solving yet, so algorithm evaluation criteria include both time efficiency and solution quality; but traditional matching algorithm is only concerned with matching time.
2. The flexibility and complexity of the PMWL problem definition are reflected in the pattern, therefore, compared with traditional matching, PMWL pay more attention to the description of the pattern information. Pattern characteristics are extremely associated with the solution of PMWL problem.

Next, we will give the representative algorithms for solving PMWL problem, and detailed description of their design ideas from the perspective of data structure and matching strategy.

### 3.1. The SAIL Algorithm

Description of SAIL Algorithm (Chen et al., 2006):

*Input:* A text  $T = t_0t_1\dots t_{n-1}$ , a pattern  $P = p_0p_1\dots p_{m-1}$ , local constraints  $g_i = g(N_i, M_i)$ , global constraints  $[minLen, maxLen]$ .

*Output:* Occurrences of  $P$  in  $T$  satisfying the constraints.

The Steps of the algorithm:

1. *Location:* ① Search position  $i$  where  $t[i] = p[m-1]$ , and locate position  $k$  where  $t[k] = p[0]$  by considering the global constraint. ② Cut out a substring in  $T$  from  $t[k]$  to  $t[i]$  named  $T'$ . ③ Build the table with the row and column according to  $T'$  and  $P$ .
2. *Forward:* Scan the table forward, and mark all the positions satisfying the local and global constraints. They are the potential matching positions.
3. *Backward:* Scan the table backward, and select the *left-most* position in the marked cells every row that compose an occurrence. Then mark them used.

Generally, SAIL starts from the beginning of  $T$  to search position  $i$  where  $t[i] = p[m-1]$ . After that, SAIL conducts two phases, the *Forward* phase and the *Backward* phase. In the *Forward* phase, SAIL determines whether there is a potential matching occurrence by using a search table. Afterwards, if a potential matching occurrence can be determined, *Backward* phase is triggered out to output an optimal occurrence by using the *left-most* strategy.



A running example for SAIL:

In this subsection, we show how SAIL works with a running example where  $P$ ,  $T$  and constraints are given as follows.

	0	1	2	3	4	5	6	7	8	9
$T$	t	t	a	a	g	g	c	c	c	c
$P$	a $\notin$ [0,1]g $\notin$ [0,1]c $\notin$ [0,1]c, $minLen = 6$ , $maxLen = 7$									

**Table 2.** A running example for SAIL

- Step 1.** Scan the  $P[m-1]$ , that is the letter ‘c’, in  $T$  from left to right. The first matching position is 6, and then SAIL enters the *Location* phase. Use the global constraint [6, 7] to locate  $P[0]$ ’s position, that is the letter ‘a’. We get the scanning range is [6-6, 7-6]. However there are no matching in [0, 1]. Then SAIL move on.
- Step 2.** The second matching position is 7, and we can locate  $P[0]$  in position 2. Then we get the substring “aaggcc” from  $T$ . In this way global constraint is satisfied.
- Step 3.** Build a 4x6 table. The row stand for character in  $P$ , and the column is the substring. Then set the position  $pos[3][5]$ ,  $pos[0][0]$  and  $pos[0][1]$  to 1.
- Step 4.** Enter *Forward* and set all the positions to 1 in the table, which satisfy the local constraints.

Positions in <i>Text</i>						
Positions in <i>Table</i>						
a						
g						
c						
c						

**Table 3.** The constructed search table  $pos[j][i-start]$  when  $P[m-1]$  is 7

- Step 5.** Enter *Backward* and select the left-most one from the marked positions in each row, and they are highlighted. In this way, we will get an occurrence {2, 4, 6, 7} and mark the four positions used.
- Step 6.** Go on to execute *Location* and get the third matching position is 8, then we can build the table below. Notice the positions of 6, 7 have been used. Under the *one-off* condition, all used positions (marked as \* in Table) of  $T$  are never considered for further matching again. If the *one-off* condition is not considered, SAIL will get another two occurrences {2, 4, 6, 8} and {3, 5, 7, 9}. Then the *Forward* phase returns false, and SAIL go back to *Location*.
- Step 7.** In position 9, the *Forward* also returns false. Finally, SAIL output only one occurrence {2, 4, 6, 7}.

Positions in <i>Text</i>						
Positions in <i>Table</i>						
		*		*	*	
a						
g						
c						
c						

**Table 4.** The constructed search table  $\text{pos}[j][i\text{-start}]$  when  $P[m-1]$  is 8

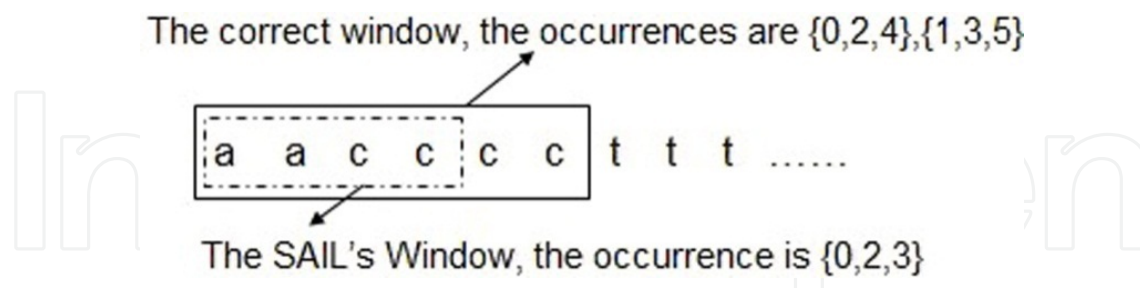
The time complexity and completeness analysis:

$O(\text{SAIL}) = O(n + klm g)$  where  $n$  is the length of  $T$ ,  $k$  is the frequency of  $P$ 's last letter occurring in  $T$ ,  $l$  is the user-specified maximum length for each matching substring,  $m$  is the length of  $P$ , and  $g$  is the maximum *gap* of wildcards in  $P$ .

Two important issues, *Online searching* and *Optimization*, are taken into consideration to design SAIL. As for optimization, under the *one-off* condition, SAIL determines which occurrence is an optimal one if multiple occurrences end at a  $P[m-1]$ 's position by applying the *left-most* strategy. As a heuristic algorithm, SAIL utilizes a kind of greedy strategy to select a set of occurrences; consequently, SAIL may obtain locally optimal solution which lead to losing occurrences in offline searching.

Form the above example, we can know that a complete occurrence set for text  $T$  is  $\{\{2, 4, 6, 8\}, \{3, 5, 7, 9\}\}$ , but SAIL's output is  $\{\{2, 4, 6, 7\}\}$ .

We believe that the SAIL's data structure is based on the sliding window, the *Location* also uses the *left-most* strategy, so it is possible to lose occurrences, for instance:



**Figure 2.** A sliding window in SAIL

Obviously, in the above example, SAIL loses occurrences in offline condition because of the selection of character  $c$ 's matching position. For further observation, it is not difficult to find that character  $c$  appears in the pattern twice. If pattern is  $a c[0,1] g c[0,1] c$ , SAIL will get a complete occurrence set, that is,  $\{\{2, 4, 6\}, \{3, 5, 7\}\}$ . Further experiments show that, the recurring appearances of pattern characters influence the quality of matching occurrences obtained by the algorithm. In next part, we will analyze the completeness of PMWL based on pattern features.

### 3.2. The RSAIL Algorithm

Description of RSAIL Algorithm (Wang et al., 2010):

**Definition 6** Given a pattern  $P$ , if there are letters  $p[i] = p[j]$  where  $0 \leq i \leq m-1, 0 \leq j \leq m-1$ ,  $P$  is called a **pattern with Recurring characters**, and **R pattern** in short, such as  $a\{0,1\}c\{0,1\}c\{0,1\}t$ .

**Definition 7** Given a pattern  $P$ , if all the letters in  $P$  are different,  $P$  is called a **pattern with No-Recurring characters** and **NR Pattern** for brevity, such as  $a\{0,1\}c\{0,1\}g\{0,1\}t$ .

**Definition 8** Given a pattern  $P$ , if there is a position  $i$  such that  $p[i] = p[i+1] = \dots = p[m-1]$  where  $1 \leq i < m-1$ ,  $P$  is called a pattern with recurring tail characters and **RT pattern** in short. Such as  $a\{0,1\}c\{0,1\}c$ . As we can see, the RT pattern is a special form of the R pattern.

From the above discussion, in the research of Chen et al., since they only concern about the on-line situation, their proof of SAIL's completeness is incomplete, which is only suitable for the on-line situation. What is more, it ignores the interaction between different occurrences.

We find that SAIL satisfies the completeness under a certain restriction, i.e. the pattern with no-recurring character (NR pattern), such as  $a\{0,1\}t\{0,1\}g\{0,1\}c\{0,1\}$ . The concept of NR pattern has practical significance, for example, in text mining, where the text is a sequence of words, the NR pattern reflects the semantic relation between words.

We utilize the symmetry to scan the text and the pattern. Then convert an RT pattern into an R pattern.

1. According to the characteristic of  $P$ , if it is not an RT pattern, we directly call SAIL; otherwise go to (2);
2. Reverse  $T$  and  $P$ , respectively get  $T', P'$ ;
3. Call SAIL, and obtain the occurrences of  $P'$  in  $T'$ ;
4. Obtain the occurrences of  $P$  in  $T$  by coordinate transformation of the obtained solution.

Obviously, since the time of the identification of pattern's characteristics is linear,  $O(\text{RSAIL}) = O(\text{SAIL})$ .

Experiments and Analysis:

We will give a set of experiments to illustrate two problems:

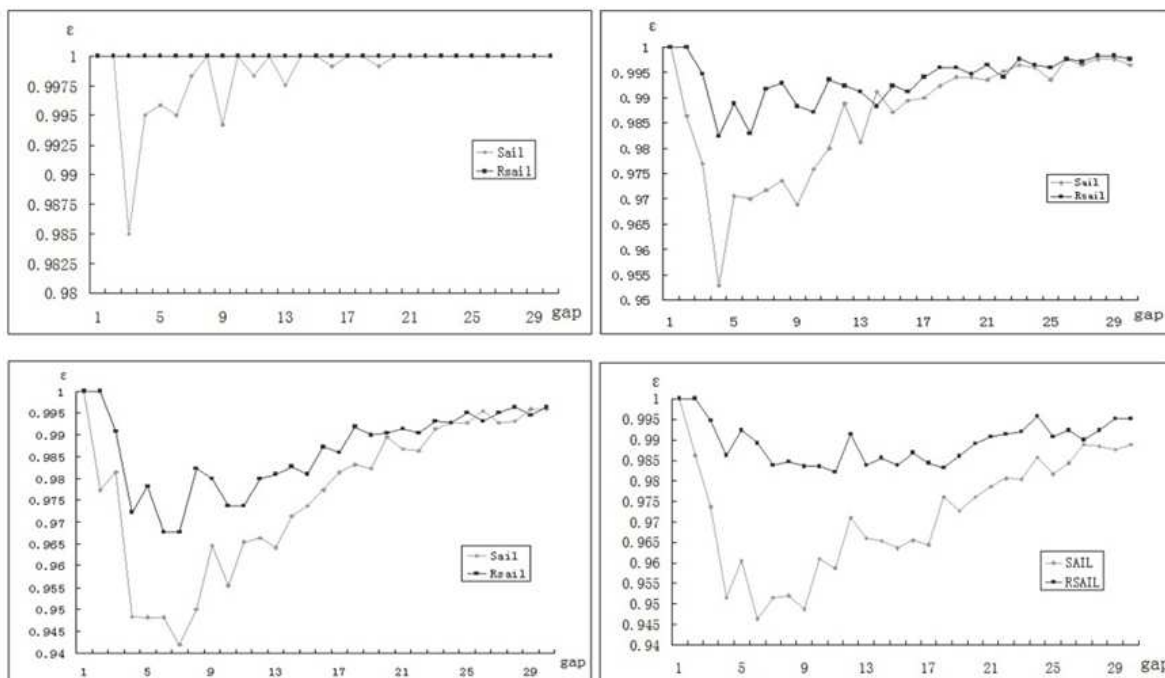
1. Analysis of the complete extent of the SAIL algorithm;
2. The comparison of the complete extent of RSAIL and SAIL algorithm;

Considering there is no algorithm can obtain the completeness occurrences of PMWL problem in polynomial time, we have developed a text generator (Xie et al., 2010) to generate experimental text, by which way we can know the completeness occurrences in order to analyze the complete extent of algorithm. In addition, the patterns used in these experiments are all RT patterns.

	Experiment1	Experiment 2	Experiment 3	Experiment 4
Size of alphabet $\Sigma$	4	4	4	7
Length of pattern $m$	3	4	5	5
$gap$	0~30	0~30	0~30	0~30

**Table 5.** The parameters in the experiments.

The experimental results and analysis:



**Figure 3.** The approximate ratio experimental results of RSAIL and SAIL

From the above images, SAIL itself is already a near-complete algorithm, in the above graphs, the average approximation ratio of SAIL is higher than 0.94; For the RT patterns, the completeness of RSAIL is better than SAIL in different  $\Sigma$ ,  $m$  and  $gap$ . Thus, not only a revised algorithm is obtained, the SAIL's deficiency on handling RT patterns is also proved from another aspect.

### 3.3. The BPBM Algorithm

Description of BPBM Algorithm (Guo et al., 2011):

Like the SAIL algorithm, the BPBM algorithm also focuses on pattern matching in online sequential text with both flexible  $gap$  constraints by user's specification and the *one-off* condition. BPBM is based on bit-parallel technology to simulate the matching process and adopt two nondeterministic finite state automatons (NFAs). One is a search mechanism to identify all pattern  $P$ 's suffix, and another one is a security window transition mechanism which accelerates the scanning process by dropping useless sequences in text.

BPBM has following characteristics:

1. BPBM also uses the *left-most* strategy to obtain the maximal occurrence of pattern in text, and return all these matching position sequences. This algorithm combines bit-parallel technology with nondeterministic finite state automata. It also simplifies the calculation of shift distance of the security window transition, which gets good results. BPBM inherits the advantage of BM algorithm to skip some of characters in text, which conducts the algorithm with a sub linear average time complexity. Therefore, the time complexity of BPBM is lower compared to SAIL.
2. Compared with Gaps-Shift-And algorithm and Gaps-BNDM algorithm, since they are all based on bit-parallel technology, their time performances are equal, however, because BPBM improves the formula of  $\varepsilon$ -transition in matching process, BPBM is fit for all patterns. In addition, BPBM returns a concrete set of matching position sequence, which makes it more applicable.
3. Compared with SAIL, SAIL applies two-dimensional table as data structure, but BPBM is based on bit-parallel technology. Since the difference in data structure, BPBM has a better time performance. The similarity between these two algorithms is they all utilize the *left-most* strategy, therefore, they are all heuristic algorithm with greedy strategy. In addition, the matching occurrences of these two algorithms are same and both incomplete.

### 3.4. The SBO Algorithm

Description of BPBM Algorithm (Wu et al., 2011):

Wu et al. propose a new nonlinear structure called *Nettree* to deal with pattern matching with flexible constraints of wildcards. A *Nettree* is a kind of directed acyclic graph (DAG) with edge labels. They apply a heuristic algorithm to select better occurrence (SBO). In this algorithm, they use two strategies: Strategy of Greedy-Search Parent, SGSP and Strategy of *right-most* Parent, SRMP to two occurrences of the same leaf, and select the better one as occurrence. The core idea of SGSP is finding an approximately optimal parent (AOP) of current node in each step; while the core idea of SRMP is finding the right-most parent node of current node in each step.

In *off-line* conditions, owing to its heuristic strategy, SBO can obtain more occurrences than SAIL and BPBM in most cases, but it is still incompleteness. However, the time complexity of SBO algorithm is  $O(gap * n * (n + m^2))$  which is nonlinear of length of text. Further more, experiments show that, in general, SBO indeed consumes more time than SAIL. In SBO, the improvement of solution's quality is relying on using heuristic strategies repeatedly, and this also consumes a lot of time. Therefore, we need consider the balance between completeness and time efficiency of algorithm. What is more, SAIL originally is not designed for *off-line* condition, as an *on-line* algorithm, it only has current information, so when applying it to *off-line* matching it will definitely be imperfect. But SBO uses global information to search occurrences, it also uses heuristic strategies to search on solution space. With the improvement of occurrences' quality, there are two problems: more

information need more place to store, so the space complexity of SBO is  $O(\text{gap} * m * n)$ , the next one is more information means more calculations thus consuming more time.

### 3.5. Other algorithms

In many literatures, similar problems are defined and various algorithms are put out to solve certain problems. Morgante, et al. (Morgante, et al., 2004) described a structured model, which can be considered as ‘compound patterns’ made of a list of simple motifs and a list of intervals that specify at what distances adjacent motifs should occur. They gave a detailed description of the biological background of the problem definition. For example, many retrotransposons belonging to the Ty1-copia group contain a match of  $\text{MT}\phi[115,136]\text{MTNTAYGG}\phi[121,151]\text{GTNGAYGAY}$ , which consists of three patterns and two intervals. As the paper pointed out, structured motifs are called classes of Characters and Bounded Gaps (CBG) expressions in Navarro and Raffinot, but use of these expressions is quite different: the underlying motivation for CBG expressions is searching in database like PROSITE and a sequence of this kind is usually not very long, while structured motifs can be very long since gaps may span many letters. As we can see, the concept of CBG and structured motifs are all have practical meaning. Because of the different application background, they design different algorithms to solve their problems. From the application point, this paper also considered a problem of q-approximation match which means just finding partial motifs in the sequence. In this paper, they proposed a two-step procedure which is used in many algorithms for PMWL: firstly, finding the occurrences of all the component patterns; secondly, combining the occurrences that satisfy the distance constraints into a structured motif. For step two, they gave a detailed algorithm to build a directed acyclic graph according to the positions of the component patterns and interval constraints. Then they discussed how to output all the occurrences in detail. In (Rahman et al., 2006), the definition of their problem likes SAIL, but they don’t consider global constraints and the *one-off* searching. In addition, just like paper (Chen et al., 2006), the local constraints exist between two substrings, while in SAIL, exist between any two consecutive letters. Certainly, a single character is a substring, but in this paper, all these substrings are used to build an AC automaton. It is not efficient to build a Trie structure over a set of single letters. This paper also used a two-step procedure: firstly using AC automaton to get occurrences of each sub-patterns in orders and combine them. They built an implicit graph, in which vertices are partitioned into several sets in order according to the corresponding sub-pattern and edges between two consecutive sets means two positions in these two consecutive sets fit corresponding local constraints. To output all  $P$  in  $T$ , we have to enumerate all possible paths in the implicit directed graph which length is the number of sub-patterns in the pattern. Morgante, et al. (Morgante, et al., 2004) applied a revised depth first searching algorithm. Philip Bille et al. (Bille et al., 2010) defined a concept named variable length *gap* (VLG) which is a pattern formed by a sequence of strings and variable length gaps. Obviously, this definition is almost the same with above works. Unlike Rahman’s work, although this paper also applies AC automaton, it maintains a sorted list containing the ranges defined by previously reported relevant occurrences, and naturally it



uses the *left-most* strategy to count an occurrence as soon as it appears. Haapasalo et al. (Haapasalo et al., 2011) extended the usual dictionary matching problem to the case in which patterns may include single wildcards, or wildcard strings of variable length with fixed or unlimited upper bound. And their algorithm is designed for *on-line* matching: the text is scanned only once, and the matches for all patterns are reported at the point of occurrence. Firstly, they constructed an AC PMA (pattern matching automaton) with output tuples identifying the keywords of the patterns to be matched. The idea in their algorithm is that they recognize keywords by the PMA and check whether or not a newly found keyword forms a continuation of a pattern prefix found thus far.

3.6. Discussion

Because of the complexity of the PMWL problem definition, we believe that the matching occurrence of PMWL problem is with high degrees of freedom. In traditional matching problem with fixed-length wildcard, the positions of each match in the same set of the matching occurrence are relatively fixed to each other. Therefore, to determine the position of any one character, a set of matching occurrences have been identified. We believe that matching of each character in above problem has a strong correlation. For instance, in pattern  $a\epsilon g\epsilon\epsilon c$ ,  $p[2] - p[1] = 1$ ,  $p[3] - p[2] = 2$ . However, for pattern in PWML, matching positions of adjacent characters are bounded by local constraints, which means matching of each character has a weak correlation, and to determine the positions of all characters, a set of matching occurrences could have been identified, that is, freedom degree of matching increases. This is an important factor leads to the complexity of the PMWL problem, which greatly increases the difficulties of searching process in matching algorithms. In order to get a complete solution, a lot of backtracking operation are required, making it difficult to be completed in polynomial time, therefore, almost all PMWL algorithms use greedy strategies in matching process. This is destined to incomplete results. However, on the other hand, although the above algorithms are not complete, we find that when length of pattern is shorter than 6, the approximation ratio of these algorithms are more than 0.9. Consequently, the next work can be considered form two aspects: 1, based on SAIL algorithm etc, improving the time efficiency, like BPBM; 2, designing algorithm for PMWL under certain conditions, such as RSAIL’s work for RT pattern. We believe that the pattern features, data structures and matching strategies will continue to be the center for PMWL algorithm design.

	SAIL	RSAIL	BPBM	SBO
Matching strategy	<i>left-most</i> (greedy strategy)	<i>left-most, right-most</i> (greedy strategy)	<i>left-most</i> (greedy strategy)	<i>SGSP, SRMP</i> (greedy strategy)
Data structure	Sliding window	Sliding window	Bit-parallel	Nettree
Time consumption	All polynomial time and SBO > RSAIL = SAIL > BPBM			
Completeness	All incompleteness, in general SBO > RSAIL > SAIL = BPBM			

**Table 6.** The strategy, structure, time consumption and completeness of PMWL methods

## 4. Analysis of PMWL based on pattern features

In the traditional matching problems, how to search pattern in text much faster as well as the correlation between pattern and text are paid more attention. But the characteristics of the pattern itself are paid little direct attention, because traditional matching problems can always have complete occurrences. The main characteristic of PMWL is with flexible wildcards, which leads to a large number of candidate matching positions. And the conflict between these occurrences will cause the final output incompleteness. However, our research shows that the direct cause which impacts PMWL incompleteness is not wildcards; it is the pattern characteristics directly conduct PMWL incompleteness. This is much different from traditional matching problems.

### 4.1. The impact of the alphabet, the length of pattern and the gap on completeness

In the traditional pattern matching research, the length of pattern and the size of alphabet are key elements influencing time complexity when analyzing traditional matching problems. Taking into account PMWL problem definition, upper and lower limits of the length constraints probably affect problem solving. Especially, instead of upper and lower limits themselves, the distance between the upper and lower limits, that is *gap*, are taken into consideration. Therefore, the parameters related to the algorithm completeness may be the size of alphabet, the length of pattern and distance between the upper and lower limits, denoted as  $\Sigma$ ,  $m$ , and *gap* respectively. In this article, the approximate degree of completeness of the algorithm will be measured by approximation ratio  $\varepsilon$ . Consequently, we try to build following model:

$$\varepsilon = F(\Sigma, m, gap) \quad (1)$$

Taking into account that the size of  $\Sigma$  is determined in a specific area, for example, in bioinformatics, DNA sequences can be defined on  $\Sigma = \{a, c, g, t\}$ , the above formula can be simplified as  $\varepsilon = F(m, gap)$ . In experiment project, input text is a biology DNA sequence, so  $\Sigma = \{a, c, g, t\}$ . Then the remaining parameter values are as follows:  $gap \in [1, 30]$ ,  $m \in [3, 9]$ , consequently, there are  $30 \times 7 = 210$  groups of experiments. The aim is to find approximation ratio  $\varepsilon$ .

Firstly, pattern  $P$  is generated randomly by pattern generator according to  $\Sigma$ ,  $m$ , and *gap*. For example, when  $m = 5$ ,  $\Sigma = \{a, c, g, t\}$ ,  $gap = 2$ ,  $a\{0,2\}c\{0,2\}c\{0,2\}t\{0,2\}g$  is a qualified pattern. For simplicity, in generated patterns, each two consecutive characters have the same length constraints i.e. *gap*. Then, what needs to be done is calculating approximate ratio  $\varepsilon$  for each pattern. Since  $\varepsilon = N(\mathbf{U}_{ALG}) / N(\mathbf{U}_{opt})$ , we need to know  $N(\mathbf{U}_{opt})$ . However, it is not desirable to directly solve this from a text  $T$ , since there is no any known algorithm to obtain the completeness solution. If we use a simple brute-force, the exponential time will be need. Therefore, we have developed a text generator, which can generate text  $T$  according to  $P$  and  $N(\mathbf{U}_{ALG})$ . In addition, SAIL algorithm is currently regarded as the most representative algorithm for PMWL problem, since SAIL firstly adopts the *left-most* strategy which is

applied in different situations and technologies such as BPBM(Guo et al., 2011) algorithm and the mining algorithm MAIL(Xie et al., 2010). Based on the above analysis, we have SAIL as a research object, that is,  $N(\mathbf{U}_{ALG}) = N(\mathbf{U}_{SAIL})$ .

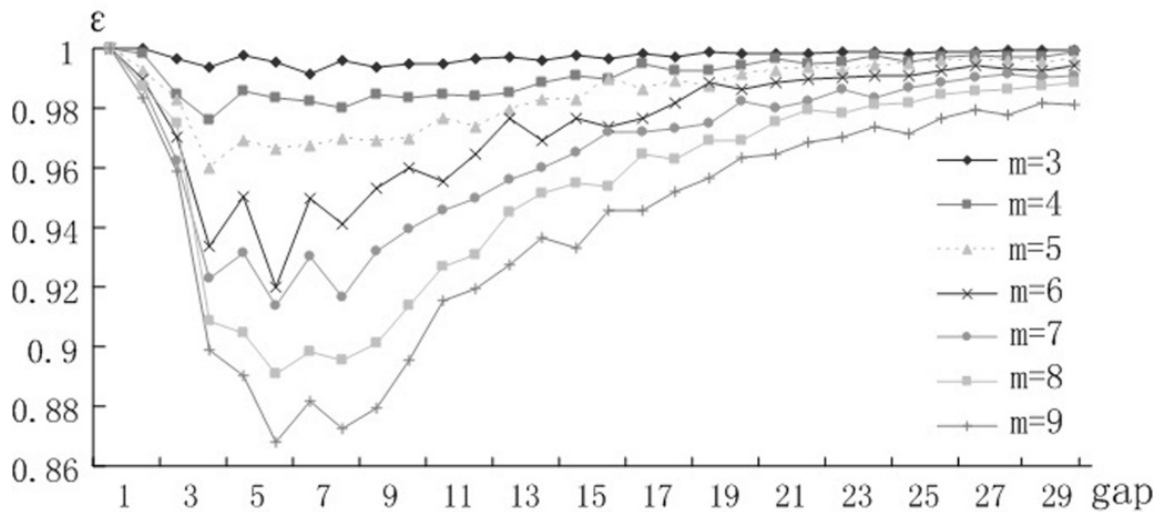
In summary, the concrete steps of the experiment are as follows:

1. For given  $\Sigma$ ,  $m$  and  $gap$ , 100 patterns  $p_i$  are generated randomly, where  $i = 1, 2, \dots, 100$ ;
2. For pattern  $p_i$ , given  $N(\mathbf{U}_{opt}) = 100$ , text length  $n = 2000$ , generate text  $T_i$ ;
3. For  $T_i$ , call SAIL algorithm to get  $N(\mathbf{U}_{SAIL})$ ;
4. Calculating  $\varepsilon_i = N(\mathbf{U}_{ALG}) / N(\mathbf{U}_{opt})$ ;
5. Calculating  $\varepsilon = \sum_{i=0}^{100} \varepsilon_i / 100$ .

	Experiment1	Experiment 2
$\Sigma$	4	7
$m$	3~9	3~9
$gap$	1~29	1~29

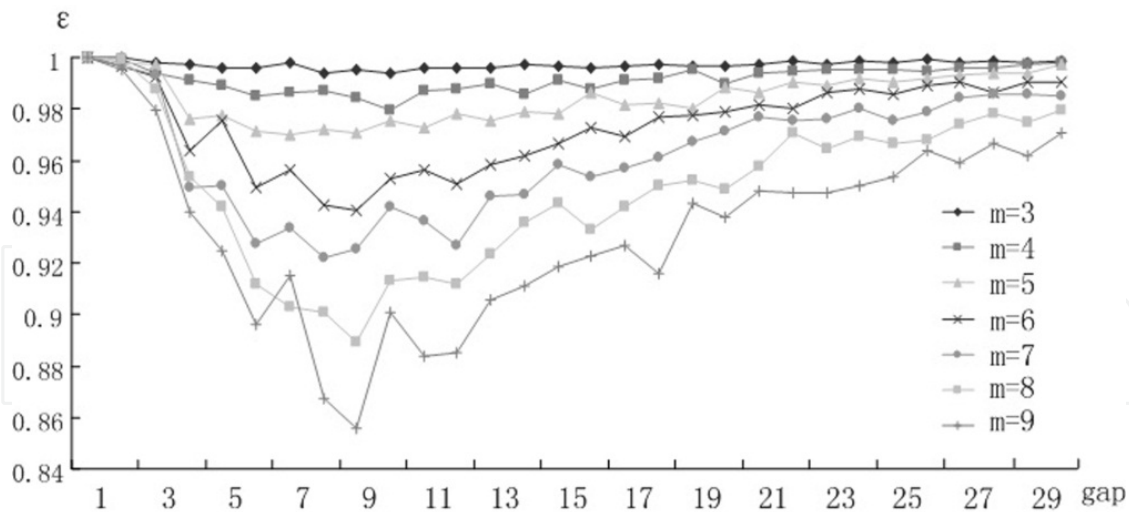
**Table 7.** Parameters in experiments for  $\varepsilon = F(gap)$

The experimental results :



**Figure 4.** Curves of  $\varepsilon = F(gap)$  in experiment 1

By the figure 4, as  $m$  increases,  $\varepsilon$  is gradually decreasing. As the  $gap$  increases, the trend of  $\varepsilon$  is decreasing first and then increases, especially when  $gap = 1$  and  $\varepsilon = 1$ , since the *left-most* strategy can obtain a complete occurrence set. With the increase of  $gap$ ,  $\varepsilon$  begin to decline because when the  $gap$  is becoming greater, the probability of matching occurrences overlap is becoming greater and the algorithm is becoming more easily to lose occurrences; when  $gap$  is sufficient, although matching occurrences are still overlap, greater  $gap$  reserve enough space for matching, making the remaining occurrences which have not yet been still have enough resources. Moreover, it is worth noting that the minimum of these curves can be reached when  $gap$  is about 7, and have nothing to do with the pattern length.



**Figure 5.** Curves of  $\varepsilon = F(\text{gap})$  in experiment 2

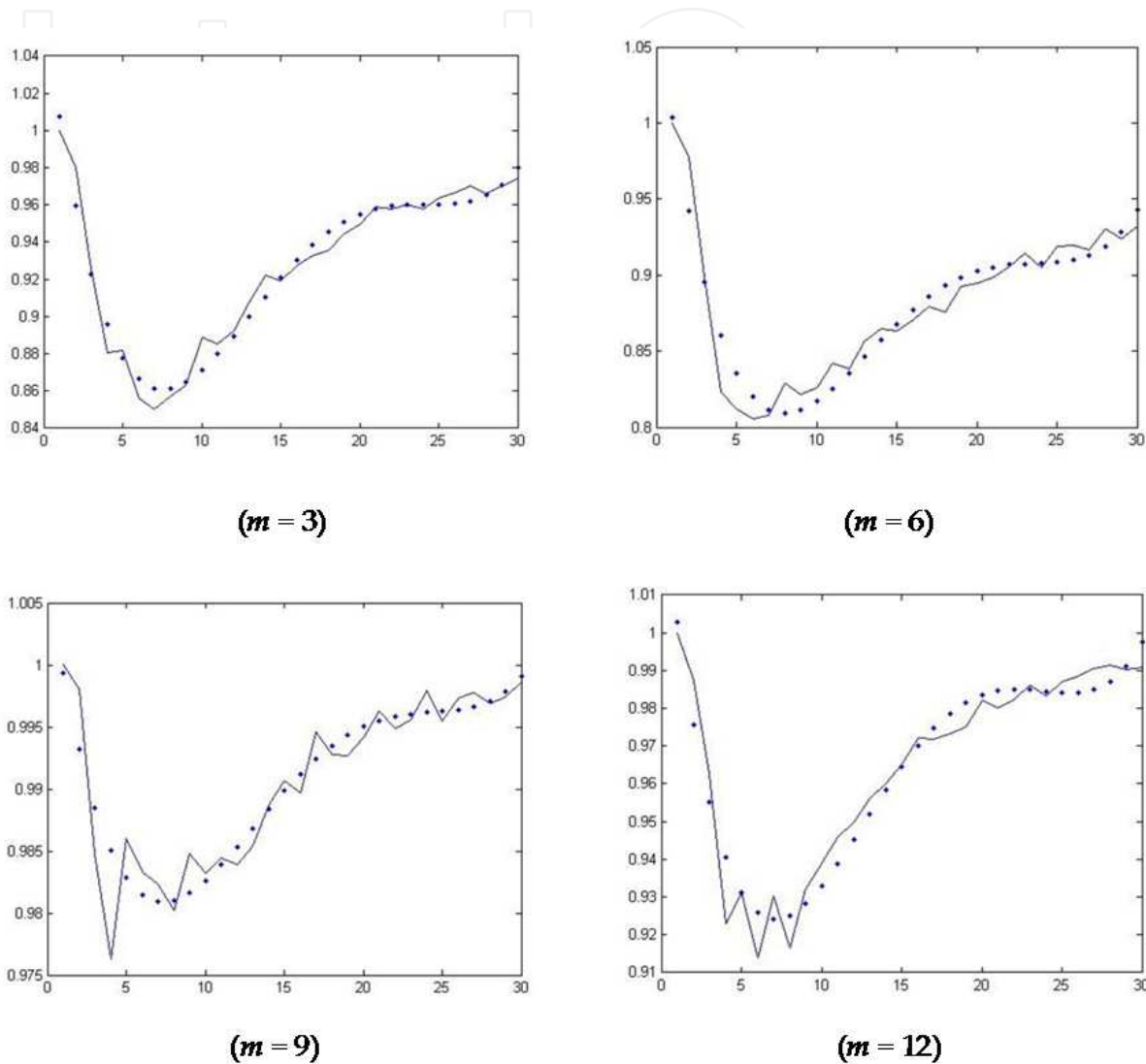
In *figure 5*, the trend of curves is the same as in *figure 4*. The difference between them is curves in *figure 5* reach the minimum when *gap* is about 9~11. It can be found that the impact of  $\Sigma$ ,  $m$ , and *gap* on the curves is that the change of *gap* determines the trend of the curve,  $m$  affects the magnitude of this change, and  $\Sigma$  makes the curve do translational move.

	A	B	C	D	E
07	1.79E-07	-1.31E-05	3.27E-04	-2.98E-03	1.0032
07	4.97E-07	-3.74E-05	9.59E-04	-8.81E-03	1.0072
07	9.95E-07	-7.37E-05	1.85E-03	-1.65E-02	1.0142
06	1.98E-06	-1.44E-04	3.51E-03	-3.05E-02	1.0276
06	2.45E-06	-1.79E-04	4.44E-03	-3.94E-02	1.038
06	3.20E-06	-2.36E-04	5.94E-03	-5.40E-02	1.061
06	3.55E-06	-2.64E-04	6.68E-03	-6.16E-02	1.0672
06	3.84E-06	-2.87E-04	7.33E-03	-6.82E-02	1.0688
1 06	4.11E-06	-3.10E-04	8.01E-03	-7.61E-02	1.0796
2 06	4.58E-06	-3.39E-04	8.61E-03	-8.13E-02	1.0809
3 06	4.79E-06	-3.53E-04	8.98E-03	-8.57E-02	1.0805
4 06	5.44E-06	-3.97E-04	9.99E-03	-9.47E-02	1.0922

**Table 8.** Parameters in mathematic model

After a series of experiments, we speculate that  $\varepsilon = A \cdot gap^4 + B \cdot gap^3 + C \cdot gap^2 + D \cdot gap + E$ , where  $A, B, C, D$  and  $E$  are parameters and for different  $m$  there are different parameters. We try to use this model to illustrate the relation between  $gap$  and approximation ratio  $\varepsilon$ .

Use this parameter table, some of illustrations for  $m = 3, 4, \dots, 14$  are listed below, where horizontal axis is the  $gap$ , vertical axis is the  $\varepsilon$ .



**Figure 6.** Model fitting

We believe this model can be used to predict the completeness of solutions given a certain pattern. For example, given  $m = 10$ ,  $\Sigma = \{a, c, g, t\}$ ,  $gap = 5$ , this model shows the prediction of approximation ratio  $\varepsilon$  of SAIL algorithm is about 0.878. Therefore, this model can be used in pattern mining showed as below.

PMWL pattern mining evaluation mechanism

*Input:* Given  $T, \Sigma, m, gap$ , support  $sup$

*Output:* pattern  $P$

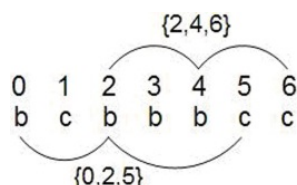
As we know, mining algorithm strategy is to learn from the strategy of matching algorithm, so PMWL pattern mining problem is naturally based on PMWL matching problem. For example, in mining algorithm MAIL (Xie et al., 2010), although a graph structure is utilized which conducts it different from SAIL; it is still based on the *left-most* strategy. As a result, they have the same degree of completeness. Therefore, our model can propose an evaluation mechanism for mining.

#### 4.2. The impact of pattern *rep* on completeness

In next part, we will put forward another important concept, named *rep*, and analyze its impact on completeness. We first give an example to illustrate the reason why this concept is needed. Given  $m = 4$ ,  $\Sigma = \{a, c, g, t\}$ ,  $gap = 2$ , the corresponding patterns maybe  $P_1 = a\text{c}[0,2]c\text{c}[0,2]g\text{c}[0,2]t$  or  $P_2 = a\text{c}[0,2]c\text{c}[0,2]c\text{c}[0,2]t$ . They have the same  $\Sigma$ ,  $m$  and  $gap$ . However, when applying SAIL or BPBM, the completeness of solutions is not the same, since for  $P_1$  algorithms can obtain complete solutions while for  $P_2$  can not.

Considering two examples below:

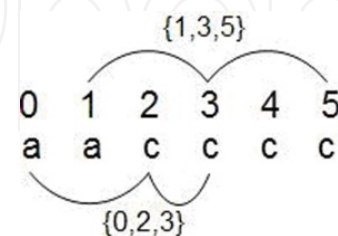
	0	1	2	3	4	5	6
<i>T</i>	b	c	b	b	b	c	c
<i>P</i>			$b\text{c}[1,2]$		$b\text{c}[1,2]c$		



**Table 9.** Example 1 for *rep* concept

A complete occurrence set of this example is  $\{\{0, 3, 5\}, \{2, 4, 6\}\}$ , the number of matching occurrences is 2. It is not difficult to find that, in SAIL algorithm, for *position* 5, the selection of position 2 as  $p[1]$ 's occurrence by the *left-most* strategy will consume the position for the next matching occurrence. We can guess that, the recurring 'b' character in this pattern affect the quality of matching occurrences.

	0	1	2	3	4	5
<i>T</i>	a	a	c	c	c	c
<i>P</i>		$a\text{c}[0,1]$		$c\text{c}[0,1]c$		



**Table 10.** Example 1 for *rep* concept

In this example, A complete occurrence set is  $\{\{0, 2, 4\}, \{1, 3, 5\}\}$ , the number of matching occurrences is 2. If we use SAIL algorithm and first obtain  $\{0, 2, 3\}$ , then we will only get this occurrence and lose  $\{0, 2, 4\}, \{1, 3, 5\}$ . Obviously, the recurring 'b' character in this pattern affects the completeness.



From above examples, the matching of recurring character in the pattern may determine the completeness of the algorithm. As a result, we consider this repeatability as an element to influence the completeness.

In order to quantify the repeatability, the concept of repeatability,  $rep$ , is proposed in this paper.

**Definition 9** Given a pattern  $P = p_0p_1\dots p_{m-1}$ , let  $f_{ij} = (p_i, p_j)$  be all binary combinations of characters in pattern  $P$

Let  $f_{ij} = \begin{cases} 0, p_i \neq p_j \\ 1, p_i = p_j \end{cases}$ , and  $rep = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} f_{ij}$ , where  $0 \leq i, j \leq m-1$ , and  $i \neq j$ . then  $rep$  is the

repeatability of characters in pattern. It shows the number of pairs of the same characters in pattern.

**Definition 10** Given occurrences  $A$  and  $S$ , if  $a[i] = s[k]$  where  $0 \leq i \leq m-1$ ,  $0 \leq k \leq m-1$ , we say  $A$  conflicts with  $S$ . For example,  $T = aacccc$ ,  $P = ac[0,1]c\{0,1\}c$ ,  $\{0,2,3\}$  conflicts with  $\{0,2,4\}$  and  $\{1,3,5\}$ . And “c” is the conflict letter.

For simplicity, global length constraint is deliberately ignored in our proof, and it does not affect the conclusion.

**LEMMA 1** Given two occurrences  $A$ ,  $S$ , if  $A$  and  $S$  come from the same occurrence set, then  $a[i] \neq s[k]$  where  $0 \leq i \leq m-1$ ,  $0 \leq k \leq m-1$ .

*Proof:* Assume  $a[i] = s[k]$ , then  $A$  conflicts with  $S$ , so they can not belong to the same set. The contradiction is achieved. Lemma 1 is proved.

**LEMMA 2** Given two occurrences  $A$  and  $S$  where  $S \in U_{SAIL}$ . If there is a conflict between  $A$  and  $S$ , and let  $a[t]$  and  $s[i]$  be the conflict positions. According to the definition 10, under the *one-off* condition,  $A$  should be discarded. Moreover, if  $i = t$ , then  $s[i] = a[i]$ ; if  $i \neq t$ ,  $s[i] < a[i]$  where  $0 \leq i \leq m-1$ . For instance,  $S = \{0, 2, 3\}$ ,  $A = \{1, 2, 4\}$ , for  $s[1] = a[1]$ , the conflict position is 1, and the other positions satisfy  $s[0] < a[0]$ ,  $s[2] < a[2]$ .

*Proof:* Assume  $s[i] > a[i]$ , then  $a[i]$  is in the left of  $s[i]$  in  $T$ . In accordance with the *left-most* strategy of SAIL, the *left-most* one prior to others is selected, which is  $a[i]$ . Due to the issue,  $S \in U_{SAIL}$ , so  $s[i]$  should be selected. The contradiction is achieved. Thus,  $s[i] \leq a[i]$ . If  $i = t$ ,  $s[i] = a[t] = a[i]$ , and if  $i \neq t$ ,  $s[i] = a[t] \neq a[i]$ . It is obvious to concluded that  $s[i] < a[i]$ .

**LEMMA 3** Given a text  $T$ , a pattern  $P$  and an occurrence  $S$ . Let  $U_{SAIL}$  be the occurrence set of SAIL. If  $S \notin U_{SAIL}$ ,  $S$  conflicts with at least one occurrence in  $U_{SAIL}$ .

*Proof:* Assume  $S$  does not conflict with any occurrence in  $U_{SAIL}$ . Then it indicates that the reason why SAIL lose  $S$  can only be the length constraint. According to the definition 4, all the occurrences satisfy the length constraint. The contradiction is achieved. So the lemma is proved.

**LEMMA 4** Let  $U_{SAIL}$  be the occurrence set of SAIL, and  $U_{opt}$  be the optimal one. Let  $N_{SAIL}$  ( $N_{opt}$ ) be the matching number in  $U_{SAIL}$  ( $U_{opt}$ ).

- (1) If  $U_{SAIL}$  is the completeness set,  $N_{SAIL} = N_{opt}$  is satisfied.
- (2) Otherwise,  $N_{SAIL} < N_{opt}$  is obtained and there is a conflict between  $U_{SAIL}$  and  $U_{opt}$ .
- (3) If the condition holds  $N_{opt} = 1$ ,  $N_{SAIL} = N_{opt}$  is obtained.
- (4) If there is no conflict,  $N_{SAIL} = N_{opt}$  is achieved.

*Proof.* It is obvious to conclude (1) is obviously true. According to the definition 5, if  $N_{SAIL} < N_{opt}$ , there is an occurrence  $S$  satisfying  $S \in U_{opt}$  and  $S \notin U_{SAIL}$ . Due to LEMMA 3,  $S$  conflicts with at least one occurrence in  $U_{SAIL}$ . That is  $U_{opt}$  is conflict with  $U_{SAIL}$ . So (2) is proved. With regard to (3), let  $S$  be the unique occurrence of  $U_{opt}$ . Assume  $N_{SAIL} < N_{opt}$ , then  $N_{SAIL} = 0$ . That is  $SAIL$  has no occurrence. In accordance with LEMMA 3,  $S$  conflicts with at least one occurrence of  $SAIL$ . But  $U_{SAIL}$  is empty, so there is no conflict. Thus the contradiction is achieved. And (3) is proved. With regard to (4), it is obvious  $N_{SAIL} \leq N_{opt}$ . We assume  $N_{SAIL} < N_{opt}$ , then there is an occurrence  $S$  satisfying  $S \in U_{opt}$  and  $S \notin U_{SAIL}$ . Due to LEMMA 3,  $S$  conflicts with at least one occurrence in  $U_{SAIL}$ . That is  $U_{opt}$  and  $U_{SAIL}$  have a conflict. The contradiction is achieved. So (4) is proved.

**LEMMA 5** Given two occurrence sets  $U_1, U_2$ , if  $U_1$  conflict with  $U_2$ , there are two sub-sets  $u_1, u_2$  with a conflict where  $u_1 \subseteq U_1, u_2 \subseteq U_2$ .

*Proof.* Assume there is no sub-sets with a conflict. All the matching positions of  $U_1$  and  $U_2$  have no conflict. According to definition 10,  $U_1$  and  $U_2$  have no conflict and satisfy the *one-off* condition. The contradiction is achieved. Lemma 5 is proved.

**LEMMA 6** Given two occurrence sets  $U_1, U_2$ ,  $U_2$  is  $U_{opt}$ . If there is a conflict between  $U_1$  and  $U_2$ , and  $N(U_1) < N(U_2)$ , there are two subsets  $u_1, u_2$  where  $u_1 \subseteq U_1, u_2 \subseteq U_2$ ,  $u_1$  is conflict with  $u_2$  and  $N(u_1) < N(u_2)$ .

*Proof.* In accordance with LEMMA 5, there are subsets  $u_1, u_2$  where  $u_1 \subseteq U_1, u_2 \subseteq U_2$  with conflict. Let  $U_1 = u_{11} \cup u_{12} \cup \dots \cup u_{1n}$ ,  $U_2 = u_{21} \cup u_{22} \cup \dots \cup u_{2m}$ , and  $u_{1i}, u_{2j}$  are arbitrary subsets where  $u_{1i} \subseteq U_1, u_{2j} \subseteq U_2, 1 \leq i \leq n, 1 \leq j \leq m$ . We discuss in three conditions: ①  $u_{1i}, u_{2j}$  have no conflict and do not satisfy  $N(u_{1i}) < N(u_{2j})$ , then  $U_1, U_2$  have no conflict and  $N(U_1) = N(U_2)$ , the contradiction is achieved. ②  $u_{1i}, u_{2j}$  have a conflict and do not satisfy  $N(u_{1i}) < N(u_{2j})$ , then  $U_1, U_2$  have no conflict, the contradiction is achieved. ③  $u_{1i}, u_{2j}$  have no conflict and satisfy  $N(u_{1i}) < N(u_{2j})$ , then  $N(U_1) = N(U_2)$ , the contradiction is achieved. So  $u_{1i}, u_{2j}$  have a conflict and satisfy  $N(u_{1i}) < N(u_{2j})$ . Lemma 6 is proved.

**THEOREM 1** Given a text  $T$ , a pattern  $P$ , if  $SAIL$  is incomplete,  $P$  must be R pattern.

*Proof.* Let  $U_{SAIL}$  be the occurrence set of  $SAIL$ ,  $U_{opt}$  is the completeness set,  $N_{SAIL}$  is the matching number of  $SAIL$ , and  $N_{opt}$  is the complete matching number. Consider the  $SAIL$  is incompleteness, according to LEMMA 4,  $N_{SAIL} < N_{opt}$ , and  $U_{SAIL}$  conflicts with  $U_{opt}$ . Due to LEMMA 6, we get two subsets  $u_1, u_2$  with conflict, which are satisfying  $N(u_1) < N(u_2)$  where  $u_1 \subseteq U_{SAIL}, u_2 \subseteq U_{opt}$ . Without loss of generality, let  $N(u_1) = 1, N(u_2) = 2$ . Set  $u_1 = \{S\}, u_2 = \{A, B\}$ , that is  $S \in U_{SAIL}, A \in U_{opt}, B \in U_{opt}$ . Let:

$$T = t[0], t[1] \dots t[i] \dots t[n-1], t[i] \text{ is stand for the } i\text{th letter in } T \text{ where } i = 0, 1, 2, \dots, n-1$$

$P = p[0], p[1] \dots p[i] \dots p[m-1]$ ,  $p[i]$  is stand for the  $i$ th letter in  $P$  where  $i = 0, 1, 2, \dots, m-1$

$A = a[0], a[1] \dots a[u] \dots a[m-1]$ ,  $a[i]$  is stand for the  $i$ th character matching position of occurrence  $A$  where  $i = 0, 1, 2, \dots, m-1$

$B = b[0], b[1] \dots b[w] \dots b[m-1]$ , another occurrence.

$S = s[0], s[1] \dots s[i] \dots s[k] \dots s[m-1]$ , another occurrence.

Let  $a[u]$ ,  $b[w]$  be the positions in  $A$ ,  $B$ , which conflict with  $s[i]$ ,  $s[k]$  in  $S$  separately. We assume other positions in  $A$  and  $B$  do not conflict with the occurrences in  $U_{SAIL}$ .  $\therefore a[u] = s[i]$ ,  $b[w] = s[k]$   $\therefore t[a[u]] = t[s[i]]$ ,  $t[b[w]] = t[s[k]]$   $\therefore$  According to the definition 4,  $t[a[u]] = p[u]$ ,  $t[s[i]] = p[i]$ ,  $t[b[w]] = p[w]$ ,  $t[s[k]] = p[k]$   $\therefore p[u] = p[i]$ ,  $p[w] = p[k]$

It would be discussed in the following two cases:

①  $u \neq i$  or  $w \neq k$

②  $u = i$  and  $w = k$

For ①, when if  $u \neq i$ ,  $\therefore p[u] = p[i]$   $\therefore$  There are two of the same letters from different positions in  $P$ .  $\therefore$  According to definition 6,  $P$  is an R pattern. For the case of  $w \neq k$ , similarly, it can be proved.

Then we will prove the other condition is impossible, and conclude  $P$  is R pattern.

For ②, we obtain  $a[u] = s[i] = s[u]$ ,  $b[w] = s[k] = s[w]$ . There is  $u \neq w$ .  $\therefore$  Assume  $u = w$ , then  $u = i = w = k$   $\therefore a[u] = s[i] = s[k] = b[w]$ . Consider  $A$ ,  $B$  belong to the same occurrence set, which contradicts with LEMMA 1  $\therefore u \neq w$ . Without loss of generality, let  $u < w$ , according to LEMMA 2  $\therefore$  SAIL adopts the *left-most* strategy, and  $S \in U_{SAIL}, A, B \notin U_{SAIL} \therefore s[u] < b[u]$ ,  $s[w] < a[w]$   $\therefore a[u] = s[u]$ ,  $b[w] = s[w]$

$$\therefore a[u] < b[u], b[w] < a[w] \quad (1)$$

And  $\therefore u < w$ , we can obtain  $b[u] < b[w]$

$A = \dots a[u] \dots a[w] \dots$

$B = \dots b[u] \dots b[w] \dots$

The occurrence  $\{b_0, b_1, \dots, b_u, \dots, a_w, \dots, a_{m-1}\}$  can be considered as  $\{\{b_0, b_1, \dots, b_u\}, \{b_u, \dots, a_w\}, \{a_w, \dots, a_{m-1}\}\}$ .

According to the definition 4,  $\{a_0, a_1, \dots, a_u, \dots, a_w, \dots, a_{m-1}\}$  and  $\{b_0, b_1, \dots, b_u, \dots, b_w, \dots, b_{m-1}\}$  satisfy the local constraints. So  $\{a_w, \dots, a_{m-1}\}$  and  $\{b_0, b_1, \dots, b_u\}$  satisfy the local constraints.

Due to  $\{b_u, \dots, a_w\}$ , we can get  $\{b_u, b_{u+1}, \dots, a_{w-1}, a_w\}$ .

From the equation (1),  $a[u] < b[u]$ ,  $b[w] < a[w]$ , and according to the definition 4:

$$b[i] < b[i+1], a[i] < a[i+1] \text{ where } u \leq i \leq w-1 \quad (2)$$

There is a  $t$  satisfying:

$$b[t+1] < a[t+1] \text{ and } a[t] < b[t] \text{ where } u \leq t \leq w-1 \quad (3)$$

$$A = \dots a[u] \dots a[t] \dots a[t+1] \dots a[w] \dots$$

$$B = \dots b[u] \dots b[t] \dots b[t+1] \dots b[w] \dots$$

Assume there is no  $t$  satisfying the condition, consider  $b[t] \neq a[t]$  where  $u \leq t \leq w$ . Then due to any  $t$  there is  $a[t+1] < b[t+1]$  or  $a[t] > b[t]$  where  $u \leq t \leq w-1$ . Consider  $a[u] < b[u]$ , there is  $a[u+1] < b[u+1]$ . Due to  $a[u+k] < b[u+k]$ , we can obtain  $a[u+k+1] < b[u+k+1]$  where  $0 \leq k \leq w-u-1$ . Then we can induce  $a[i] < b[i]$  where  $u \leq i \leq w$ . It contradicts  $b[w] < a[w]$ , so the assume is incorrect. Due to equation (2) and (3),  $a[t] < b[t] < b[t+1] < a[t+1]$  where  $u \leq t \leq w-1$ .

$$b[t+1] - b[t] < a[t+1] - b[t] < a[t+1] - a[t] \quad (4)$$

That is  $a[t]$  and  $b[t-1]$  satisfy the local constraints. In this way,  $\{b_u, b_{u+1}, \dots, a_{w-1}, a_w\}$  can be considered as  $\{\{b_u, b_{u+1}, \dots, b_{t-1}\}, \{b_t, a_{t+1}\}, \{a_{t+2}, \dots, a_{w-1}, a_w\}\}$ . In accordance with definition 4,  $\{b_u, b_{u+1}, \dots, b_{t-1}\}, \{a_{t+2}, \dots, a_{w-1}, a_w\}$  satisfy the local constraints.  $\therefore \{b_u, b_{u+1}, \dots, a_{w-1}, a_w\}$  satisfy the local constraints.  $\therefore$  From the above analysis,  $\{b_0, b_1, \dots, b_u, \dots, a_w, \dots, a_{m-1}\}$  satisfy the local constraints.

However, according to the theorem, the other positions in A,B do not conflict with  $U_{SAIL}$  except for  $a[u]$ ,  $b[w]$ . That is,  $\{b_0, b_1, \dots, b_u, a_w, \dots, a_{m-1}\}$  satisfies the *one-off* condition.  $\therefore \{b_0, b_1, \dots, b_u, a_w, \dots, a_{m-1}\}$  is another occurrence, and does not conflict with any occurrences in  $U_{SAIL}$ . But  $U_{SAIL}$  does not include this occurrence. It contradicts with LEMMA 3. Thus, condition ② is impossible. And from the analysis of ①, under the condition of the theorem, P must be R pattern. The theorem 1 is proved.

**THEOREM 2** Given a text  $T$ , a pattern  $P$ , if  $P$  is NR pattern, then SAIL is complete.

*Proof:* It is the inverse negation of THEOREM 1. Apparently, THEOREM 2 is true.

**THEOREM 3** Given a text  $T$ , a pattern  $P$ , if  $P$  is R pattern, then SAIL is incomplete.

*Proof:* It can be concluded from the analysis and example in section 2.

**THEOREM 4** If the pattern fulfills  $gap = 0$ , SAIL is complete.

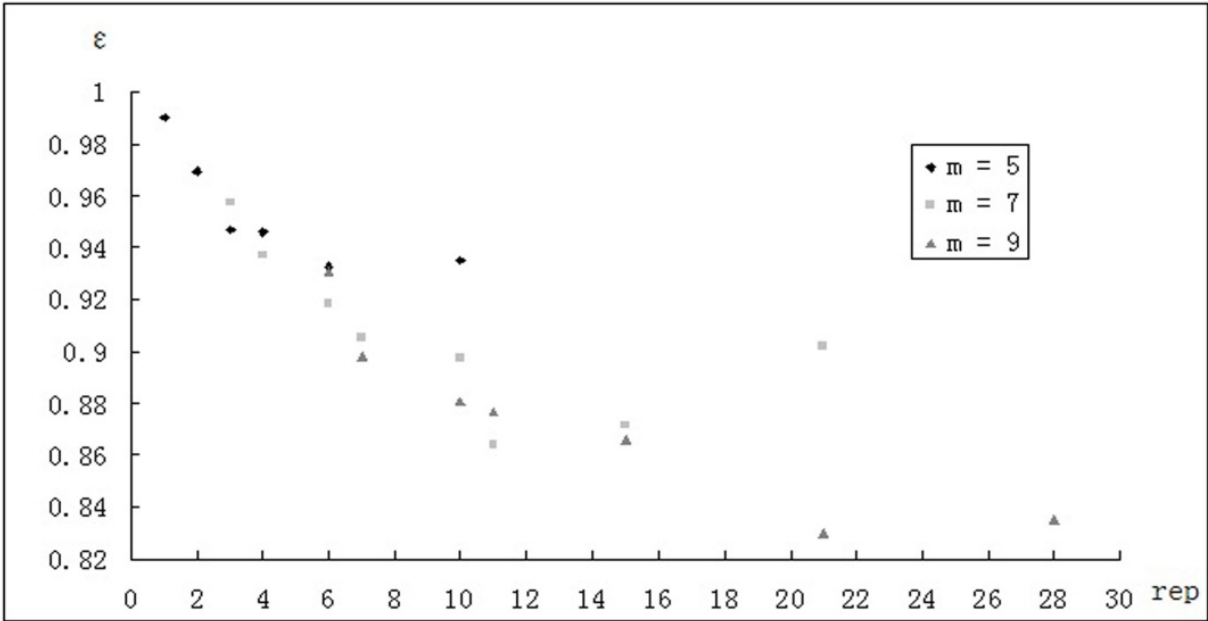
*Proof:* If  $gap = 0$ , the wildcard is a constant. For example  $a\{1,1\}c\{2,2\}c$  is converted into  $a\{1,1\}c\{2,2\}c$ . There won't be any conflict or exist seizing between occurrences. SAIL will perform complete.

**Experiment design<sup>1</sup>:**  $\Sigma = 4$ ,  $m = \{5, 7, 9\}$ ,  $gap = [0, 3]$ ,  $rep = \{0, 1, 2, 3, 4, 6, 7, 10, 11, 15, 21, 28, 35\}$ . In each set of experiments, 20 patterns are randomly generated; the final result is the average.

Analysis of experimental results: with increment of  $rep$ , the curve of approximation ratio gradually decreases, followed by a slight increase. The reason for decline is that  $rep$  lead to more nested occurrences, resulting in a greater degree of the possibility of losing occurrences; the reason for the increscent is that larger  $rep$  can cause more extreme pattern. For instance, when  $\Sigma = 4$ ,  $m = 7$ ,  $rep = 21$ , patterns like  $P_1 =$

<sup>1</sup> When  $\Sigma$  and  $m$  are determined,  $rep$  can only be some certain values, because  $rep$  has correlation with  $\Sigma$  and  $m$

$a\epsilon[0,3]a\epsilon[0,3]a\epsilon[0,3]a\epsilon[0,3]a\epsilon[0,3]a\epsilon[0,3]a$  which is difficult to find a special text containing nested occurrences of such pattern, will be produced. For  $P_1$ , the text like “aaaaaaaaaaaaa” contains nested occurrences of this pattern. Obviously, this extreme text is very rare. Therefore, under the premise of nested occurrences are not easily to be formed, the approximation ratio will be increased slightly.



**Figure 7.** The relation between  $rep$  and approximation ratio  $\epsilon$

Next we will analyze the relationship between the repeatability  $rep$  and alphabet size  $\Sigma$ , pattern length  $m$ . Original problem: a pattern which length is  $m$ , and alphabet size is  $\Sigma$ , what is the expectation of repeatability  $E(rep)$ ?

This description is equivalent to the model of ‘taking ball from the bag’ in the combination mathematics:

There is a bag of balls, and  $|\Sigma|$  kinds of colors, taking  $m$  balls from the bag with replacement, then in fetched balls, how many pairs of the same color?

$\Sigma \backslash m$	3	4	5	6	.....	$m$
3	3/3	6/3	10/3	15/3	.....	$C_m^2 / 3$
4	3/4	6/4	10/4	15/4	.....	$C_m^2 / 4$
5	3/5	6/5	10/5	15/5	.....	$C_m^2 / 5$
6	3/6	6/6	10/6	15/6	.....	$C_m^2 / 6$
.....	.....	.....	.....	.....	.....	.....
$\Sigma$	$3/ \Sigma $	$6/ \Sigma $	$10/ \Sigma $	$15/ \Sigma $	.....	$C_m^2 /  \Sigma $

**Table 11.** The relationship between  $\Sigma$ ,  $m$  and  $rep$

Finally, we can deduce:

$$E(rep) = \frac{C_m^2}{|\Sigma|} \quad (2)$$

## 5. Conclusions

As an extension of traditional matching problem, the PMWL problem has aroused more and more attention because of its unique flexibility and complexity. Based on problem definition and drawing on research idea in traditional matching problem, this article introduces SAIL, RSAIL, SBO and BPBM which are representative algorithms for PMWL in three important respects: the data structures, the matching strategies and the characteristics of pattern. The article also analyzes the pros and cons of the above algorithms from the point of quality of the solution and time complexity, and gives experimental matching results by using real DNA data. Among them, the SAIL algorithm is the first to propose the method of solving PMWL problem, it uses the sliding window structure and the representative *left-most* matching strategy. This paper finds that in short patterns, the approximation ratio of SAIL is higher than 0.9, while in longer patterns, the occurrences obtained by SAIL are of poor quality; the quality of occurrences obtained by SBO is best, but its time consumption has a non-linear relationship with the length of text; BPBM utilizes bit parallel technology to improve the efficiency of matching greatly, but also is impact by the machine word; for pattern with repeated letters in tail, RSAIL uses symmetry to improve the quality of occurrences under certain conditions, thus providing a solving idea to PMWL problem, but in longer patterns and wilder gaps, the efficiency is not obvious.

Afterwards, this article focus on relationship between approximation ratio  $\varepsilon$  and alphabet size  $\Sigma$ , pattern length  $m$ , wildcards span  $gap$  and repeatability  $rep$ . Firstly, this article proposes the model  $\varepsilon = F(\Sigma, m, gap)$ , describing the functional relationship between pattern characteristics and approximation ratio approximately; secondly, this article proves PMWL's completeness under the conditions of  $rep = 0$ ; finally, the relationship between the pattern features are also analyzed and in addition, relationship that  $E(rep) = \frac{C_m^2}{|\Sigma|}$  is proposed.

In future work, the formal description of the PMWL problem will be considered, in order to explain the complexity of the problem better, thus helping algorithm design and analysis for problem complexity.

## Author details

Haiping Wang, Taining Xiang and Xuegang Hu  
Hefei University of Technology, China



## 6. References

- Amir, A., Aumann, Y., Landau, G., Lewenstein, M. & Lewenstein, N. (2000). Pattern matching with swaps, *Journal of Algorithms*, 37(2): 247-266
- Amir, A. & Navarro, G. (2009). Parameterized matching on non-linear structures, *Information processing letters*, 109(15): 864-867
- Baeza-Yates, R. & Gonnet, G. (1992). A new approach to text searching, *Communications of the ACM*, 35(10): 74-82
- Bille, P., Gørtz, I. L., Vildhøj, H. & Wind, D. (2010). String matching with variable length gaps, *Proceedings of 17th SPIRE*, pp. 385-394
- Brudno, M., Steinkamp, R. & Morgenstern, B. (2004). The CHAOS/DIALIGN WWW server for multiple alignment of genomic sequences, *Nucleic Acids Research*, 32: 41-44
- Califf, M. E. & Mooney, R. J. (2003). Bottom-up relational learning of pattern matching rules for information extraction, *Journal of Machine Learning Research*, 4(6): 177-210
- Chen, G., Wu, X. D., Zhu, X. Q., Arslan, A. N. & He, Y. (2006). Efficient string matching with wildcards and length constraints, *Knowledge and Information Systems*, 10(4): 399-419
- Cole, J. R., Chai, B., Marsh, T. L., Farris, R. J., Wang, Q., Kulam, S. A., Chandra, D. M., McGarrell, D. M., Schmidt, T. M., Garrity, G. M. & Tiedje, J. M. (2005). The ribosomal database project(RDP-11): Sequences and tools for high-throughput rRNA analysis, *Nucleic Acids Research*, 33(1): 294-296
- Cole, R., Gottlieb, L. A. & Lewenstein, M. (2004). Dictionary matching and indexing with errors and don't cares, *Proceedings of the 36th ACM Symposium on the Theory of Computing*, ACM Press, New York, NY, USA, pp. 91-100
- Fischer, M. J. & Paterson, M. S. (1974). String matching and other products, In Karp RM(ed) *Complexity of computation*, Massachusetts Institute of Technology, Cambridge, MA, USA, vol 7, pp. 113-125
- Gusfield, D. (1997). Algorithms on strings, trees and sequences: computer science and computational biology, chapter 6, Cambridge University Press
- Guo, D., Hong, X. L., Hu, X. G., Gao, J., Liu, Y. L., Wu, G. Q. & Wu, X. D. (2011). A Bit-Parallel Algorithm for Sequential Pattern Matching with Wildcards, *Cybernetics and Systems*, 42(6): 382-401
- He, D., Wu, X. D. & Zhu, X. Q. (2007). SAIL-APPROX: An efficient on-line algorithm for approximate pattern matching with wildcards and length constraints, *IEEE International Conference on Bioinformatics and Biomedicine (BIBM'07)*, IEEE Computer Society, pp. 151-158
- He, Y., Wu, X. D., Zhu, X. Q. & Arslan, A. N. (2007). Mining Frequent Patterns with Wildcards from Biological Sequences [C], *IEEE International Conference on Information Reuse and Integration*, Las Vegas, IL, pp. 329-334

- Ji, X. N., Bailey, J. & Dong, G. Z. (2007). Mining minimal distinguishing subsequence patterns with gap constraints, *Knowledge and Information Systems*, 11(3): 259-286
- Kucherov, G. & Rusinowitch, M. (1995). Matching a set of strings with variable length don't cares, *Proceedings of the 6th Symposium on Combinatorial Pattern Matching*, Springer, Berlin Heidelberg New York, pp. 230-247
- Manber, U. & Baeza-Yates, R. (1991). An algorithm for string matching with a sequence of don't cares, *Information Processing Letters*, 37(3): 133-136
- Min, F., Wu, X. D. & Lu, Z. Y. (2009). Pattern matching with independent wildcard gaps, *Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC-2009)*, Chengdu, China, pp. 194-199
- Morgante, M., Policriti, A., Vitacolonna, N. & Zuccolo, A. (2004). Structured motifs search, *Proceedings of the 8th annual international conference on Computational molecular biology*, In print
- Muth, R. & Manber, U. (1996). Approximate multiple string search, *Combinatorial Pattern Matching*, Springer, pp. 75-86
- Muthukrishnan, S. & Krishna, P. (1994). Non-standard stringology: algorithms and complexity [C], *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing* New York, NY, USA, pp. 770-779
- "National center for biotechnology information website", [online], available: <http://www.ncbi.nlm.nih.gov/>
- Navarro, G. & Raffinot, M. (2001). Flexible pattern matching in strings: practical on-line search algorithms for texts and biological sequences, Cambridge University Press
- Rahman, M. S., Iliopoulos, C., Lee, I., Mohamed, M. & Smyth, W. F. (2006). Finding Patterns with Variable Length Gaps or Don't Cares, *Computing and Combinatorics, 12th Annual International Conference, COCOON 2006*, Taipei, Taiwan, August 15-18, Proceedings. Vol. 4112
- Sagot, M. F. & Viari, A. (1996). A Double Combinatorial Approach to Discovering Patterns in Biological Sequence, *Proceedings of the 7th Symposium on Combinatorial Pattern Matching*, Springer, pp. 186-208
- Wang, H. P., Xie, F., Hu, X. G., Li, P. P. & Wu, X. D. (2010). Pattern Matching with Flexible Wildcards and Recurring Characters, *Proceedings of 2010 IEEE International Conference on Granular Computing*, pp. 782-786
- Wu, Y. X., Wu, X. D., Jiang, H. & Min, F. (2011). A Heuristic Algorithm for MPMGOOC, *Chinese Journal of Computers*, 34(8): 1452-1462
- Xie, F., Wu, X. D., Hu, X. G., Gao, J., Guo, D., Fei, Y. L. & Ertian, H. (2010). Sequential Pattern Mining with Wildcards [C], *22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 241-247

Zhang, M. H., Kao, B., Cheung, D. W. & Yip, K. Y. (2005). Mining periodic patterns with gap requirement from sequences, *Proceedings of ACM SIGMOD*, Baltimore Maryland, pp. 623–633

IntechOpen

IntechOpen