Mining Complex Patterns across Sequences with Gap Requirements[*]

Xingquan Zhu[1, 3] **and Xindong Wu**[2]
[1]Dept. of Computer Science & Eng., Florida Atlantic University, Boca Raton, FL 33431, USA
[2]Dept. of Computer Science, University of Vermont, Burlington, VT 05405, USA
[3]Graduate University, Chinese Academy of Sciences, Beijing 100080, China
xqzhu@cse.fau.edu; xwu@cs.uvm.edu

Abstract

The recurring appearance of sequential patterns, when confined by the predefined gap requirements, often implies strong temporal correlations or trends among pattern elements. In this paper, we study the problem of mining a set of gap constrained sequential patterns across multiple sequences. Given a set of sequences S_1, S_2,., S_K constituting a single hyper-sequence S, we aim to find recurring patterns in S, say P, which may cross multiple sequences with all their matching characters in S bounded by the user specified gap constraints. Because of the combinatorial candidate explosion, traditional Apriori-based algorithms are computationally infeasible. Our research proposes a new mechanism to ensure pattern growing and pruning. When combining the pruning technique with our Gap Constrained Search (GCS) and map-based support prediction approaches, our method achieves a speed about 40 times faster than its other peers.

1 Introduction

Many real-world applications involve data characterized by continuous sequences and streams. Examples include data flows in medical ICU (Intensive Care Units), network traffic data, stock exchange rates, and DNA and protein sequences. Since real-world events rarely happen independently but rather associate with each other, to some degree, discovering structures of interest in multiple sequences provides us an effective means to explore patterns trivial in a single data stream but significant when unifying all observed data into one view. For example, the information from multiple data streams in ICU (such as the oxygen saturation, chest volume and hear rate) may indicate or predicate the state of a patient's situation, and an intelligent agent with the ability to discover knowledge from multiple sensors can automatically acquire and update its environment models [Oats & Cohen 96]. In microbiology, it is now well known that the genomes of most plants and animals contain a large quantity of repetitive DNA fragments. Examples include recurring short base pairs (BP) in protein coding DNA and repetitive DNA/RNA motifs in genomes, where recurring patterns of different lengths and types are commonly found, at both genomic and proteomic levels, to have significant biological/medical values. For example, the 10-11 BP periodicities in complete genomes reflect protein structure and DNA folding [Herzel et al. 99] and some tandem repeats are now discovered to be influential to the bacterial virulence to human [Belkum et al. 97]. Studying correlations among multiple gene sequences, their associations with environments and disease phenotypes, thus provides a means of predicting and preventing fatal diseases [Rigoutsos & Floratos 98].

Although recurring patterns convey important and useful knowledge, in reality, they rarely just reproduce and repeat themselves, but rather appear with a slight shift in the pattern letters. For example, the tandem repeats in DNA or protein sequences often involve a phase shift incurred by the insertion or deletion of a short sequence [Belkum et al. 97]. A practical solution is to allow the mining or search process to bear a certain degree of flexibility. Consider sequences in Fig. 1(a), where a pattern across three sequences repeats three times but with each appearance slightly different from the others. If we can allow that each time the pattern appears, any two of its successive pattern letters' appearances are within a range, rather than a fixed value, we may then be able to find the pattern in Fig. 1(a). The introduction of a variable period (gap) thus provides a flexible way to capture interesting patterns hidden in sequences.

Mining recurring patterns from sequences essentially relies on a counting mechanism to check patterns' occurrences. This, however, is inherently complicated by the sequential order of the underlying sequences. Considering sequence "AAAGGGTTTTCCCTTTTCCCTTTTCCCC", to find the number of complete occurrences of pattern P=AGTC (we ignore gap constraints at this stage), there are 3 × 3 combinations for "A" and "G", and 4×10+4×7+4×4 combinations for "T" and "C". So in total, there are 9×84 occurrences for AGTC. Although this number does not sound scary, considering complete occurrences, however, brings one of the most difficult challenges to our problem: the deterministic Apriori theory (the support of a pattern cannot exceed the support of any of its sub-patterns) does not hold. Considering S="AGCTTT", pattern P=AGCT

[*] This material is based upon work supported by the US National Science Foundation under Grant No. 0514819. The support of NSF of China under Grant No. 60674109 is also acknowledged.

appears three times in S, but P's subpattern AGC appears once only. Therefore, traditional Apriori-based algorithms are computationally infeasible to handle our problem.

Motivated by the above observations, we propose *MCPaS* to Mine Complex Patterns across Sequences with gap requirements. We will review related work in the next section, and state our research problem in Section 3. In Section 4, we will study pattern frequencies and propose a deterministic pruning technique for pattern mining. In Section 5, we discuss our unique Gap Constrained Search and Map-based Support Prediction processes to accelerate the pattern mining process. Based on the proposed pruning and searching techniques, we elaborate, in Section 6, the mining algorithm details. Comparative studies are reported in Section 7, followed by the concluding remarks in Section 8.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
S_1	A	.	.	.	A	.	.	.	.	A	.	.	.	.
S_2	.	e	g	.	.	.	.	e	g	.	.	e	.	g
S_3	.	.	3	.	.	.	.	.	3	.	.	.	.	3

(a)

	1	2	3	4	5	6	. . .
S_{hyb}	(A . .)	(. e .)	(. g 3)	(. . .)	(A . .)	(. . .)	. .

(b)

Figure 1. (a) Patterns across three sequences; (b) The hybrid sequence generalized from the sequences in (a)

2 Related Work

Existing research in mining patterns from data sequences can be distinguished into two categories. (1) Mining patterns frequently appearing in a certain number of (relatively short) sequences with a *Boolean count*, *i.e.*, whether a pattern occurs in the sequence or not. Traditional pattern mining in market baskets [Srikant & Agrawal 96, Pei et al. 01, Zaki 98] and Gene Motif search [Murray et al. 02] fall into this category. (2) Mining recurring patterns from long sequences such as episode mining [Yang et al. 00, Méger & Rigotti 04, Das et al. 98] and tandem repeats and base pair oscillation detection in DNA sequences [Herzel et al. 99].

Any mining process will have to rely on a counting mechanism to check patterns' frequency information, from which frequent patterns can be found. It is worth noting that the selected counting mechanism crucially impacts on the mining approach. For research efforts relying on a *Boolean count*, because a pattern's appearances are counted for only once *w.r.t.* each sequence, the deterministic Apriori theory well holds, and is therefore commonly adopted in the mining process. On the other hand, when one has to determine a pattern's actual number of occurrences in the sequences, the situation becomes complicated, simply because the mining process won't follow Apriori theory at all. Existing efforts in the field have therefore proposed confined occurrence counting, such as minimal occurrences [Mannila et al. 97], one-off occurrences [Chen et al. 05], and windowing occurrences [Mannila et al. 97]. The objective is to confine patterns' occurrences such that Apriori theory can apply.

When considering complex patterns across multiple sequences, the work relevant to our problem here comes from Oates *et al.* [96] and Mannila *et al.* [97]. Both efforts tried to search frequent episodes across sequences, as well as the prediction rules in the form of x indicating y, where x and y are the frequent episodes in the sequences. For example, after event A happens, exactly two time points later event B happens, and then exactly three time points later event C happens. Notice that this is a very restrictive constraint, and we are trying to find episodes like: after event A happens, *within* two time points event B happen, and *within* three time points later event C happens. This loose constraint leaves a great flexibility to explore useful patterns.

The problem of complex pattern mining across multiple sequences is similar to multi-dimensional sequential pattern mining [Pino et al. 01], where a normal practice is to aggregate values from all sequences to form a single sequence, as shown in Figure 1(b), so traditional sequential pattern mining methods can apply.

In the domain of DNA or protein sequences, BLAST [Altschul 90] is one of the most famous algorithms. Given a query sequence (pattern), it searches for a matching sequence from the databases, while what we are pursuing here is on mining the patterns. To find patterns, the TEIRESIAS algorithm [Rigoutsos & Floratos 98] is designed for pattern discovery from biological sequences with the number of wild-cards that can be present in the extracted patterns restricted and fixed by the users. Similar approaches such as Pratt [Jonassen 97] is also proposed to mine restricted patterns (in terms of the maximum numbers of characters and wild-cards in each pattern) from a single sequence.

3 Problem Statement

The sequences S_1,.., S_i, .., S_K from which we extract patterns are called a **hyper-sequence**, denoted by S, with each single sequence S_i called an **element sequence**. We use $S[i]$ to represent the characters at the i^{th} time point of S. Without losing the generality, we assume all element sequences share the same alphabet size, denoted by •, and |•| represents the size of •. For simplicity, we assume that all element sequences have the same lengths, denoted by L.

A wild-card (denoted by a single dot ".") is a special symbol that matches any character in •. A gap g is a sequence of wild-cards, bounded by the maximal and minimal values. The size of the gap refers to the number of wild-cards in it. We use g_N^M to represent a gap whose size is within the range $[N, M]$. We also call $W=M-N+1$, the **gap flexibility**. A **pattern,** $P=p_1g^1p_2g^2,\ldots,g^{l-1}p_l$, is a set of characters from different element sequences and gaps that begin and end with characters, where p_j is an **element pattern** which consists of letters from element sequences. An element pattern should consist of at least one character. g^j is the gap requirement between element patterns p_{j-1} and p_j. Figure 2 pictorially shows a pattern P with three element patterns. The number of element patterns in P, denoted by $|P|$, is called the length of P, *i.e.*, the wild-card symbols are not counted towards a pattern's length.

The problem of mining complex patterns across multiple sequences is to find patterns of the following form:

$$P = p_1 g_N^M p_2 g_N^M \cdots p_{l-1} g_N^M p_l \qquad (1)$$

which means that gaps between any two successive element patterns are the same, *i.e.*, $g^1 = g^2 = \ldots = g^l = g_N^M$. An occurrence of a pattern P in S is defined as a match between the characters of the element patterns and the element sequences, under the gap constraints. The occurrences are considered different, as long as the locations of one pair of the matched characters are different. For example, we consider P=ATG appears 2 times in S="ATTG", w.r.t. g_0^2, *i.e.*, S[1]S[2]S[4] and S[1]S[3]S[4]. The **support** of P in S (denoted by $sup(P)$) is the number of times P occurring in S.

Given a length-l pattern P and a length-L hybrid-sequence S, we first calculate the total number of possible occurrences of a length-l pattern in S, L_l, then we count the actual appearances of P, $sup(P)$. We consider P a frequent pattern, iff $sup(P)/L_l$ is larger than the user-specified threshold value ρ.

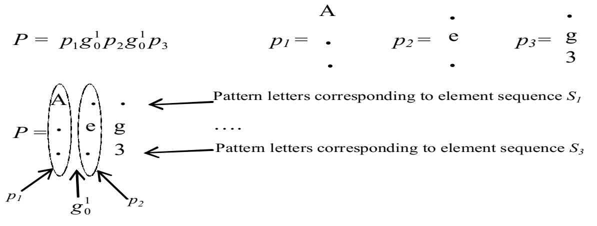

$P = p_1 g_0^1 p_2 g_0^1 p_3$
$p_1 =$ A . .
$p_2 =$. e .
$p_3 =$. g 3
Pattern letters corresponding to element sequence S_1
Pattern letters corresponding to element sequence S_3
p_1
g_0^1
p_2

Figure 2. A pattern denotation

4 Pattern Frequency & Deterministic Pruning

4.1 Pattern Frequency

Given pattern $P=p_1p_2\ldots p_l$ and its support in a length-L hyper-sequence S (with $L >> l$), sup(P), to assess *P's* frequency, we need to find L_l, the possible number of occurrences of P in S. Considering pattern P with gap g_N^M, each time P appears in S, its actual spans in S vary from $l+(l-1)N$ to $l+(l-1)M$, which correspond to the cases that whenever P appears in S, the gap between any two successive element patterns exactly equal to N and M respectively. Assuming the first element pattern p_1 matches S at $S[\delta]$, then for element pattern p_2, its valid occurrences may possibly appear in the range from $S[\delta+N]$ to $S[\delta+M]$, *i.e.*, with $W=M-N+1$ possibilities. The same situation holds for all other element patterns $p_3,\ldots p_l$. So in total, a length-l pattern P starting at S[δ] may have W^{l-1} possible occurrences in S. Assuming p_1 has possible appearances in S, the total number of possible appearances of P in S is $L_l= \cdot W^{l-1}$. Because the average span between successive element patterns is (N+M)/2+1, the average span of P in S is (l-1)·((N+M)/2+1). Then, the possible number of occurrences of p_1 equals to [L-(l-1)·((N+M)/2+1)], where [♦] means the maximal integer no larger than ♦. So the value of L_l is defined by Eq. (2).

$$L_l = [L - (l-1)(\frac{M+N}{2} + 1)] \cdot W^{l-1} \qquad (2)$$

Eq (2) holds only if (l+(l-1) ·M) • L, *i.e.*, the maximal span of the pattern is less than the length of the hyper-sequence.

4.2 Deterministic Pruning

In this subsection, we derive one theorem and two lemmas for the deterministic pruning of our mining process.

THEOREM 1. Gien a length-l pattern P and its length l-3 subpatterns Q, we have the supports of Q and P satisfy the inequality $\sup(P) \le \sup(Q)\cdot(W^3+W^2)/2$

Proof: Because Q is a length l-3 subpattern of P, denoting Q by $q_1q_2\ldots q_{l-3}$, there are four possible relationships between them: (1) $P=p_1Qp_{l-1}p_l$; (2) $P=p_1p_2Qp_l$; (3) $P=Qp_{l-2}p_{l-1}p_l$; and (4) $P=p_1p_2p_3Q$. Let's first prove that Theorem 1 is true for (1). The same proof applies to all other possibilities.

Assuming N=0 and the gap flexibility is W, the first element pattern of Q, q_1, appears at time slot $S[\delta]$. It is easy to know that p_1 has W possibilities to appear between $S[\delta-W]$ and $S[\delta-1]$. So the maximal support of p_1Q is $sup(p_1Q)=W\cdot sup(Q)$. Now assuming further that the last element pattern of Q, q_{l-3}, appears at $S[\delta+\beta]$, it is clear that p_{l-1} has W possibilities to appear at the range between $S[\delta+\beta+1]$ and $S[\delta+\beta+W]$, as shown in Figure 3. If p_{l-1} indeed appears at $S[\delta+\beta+W]$, the element pattern p_l will have W possibilities to appear between S[$\delta+\beta+W+1$] and S[$\delta+\beta+2W$]. Denoting this region by $\varphi_{l-1}^1 \subset [\alpha+\beta+W+1, \alpha+\beta+2W]$, if p_{l-1} appears at $S[\delta+\beta+W-1]$, we know that p_l may possibly appear between $S[\delta+\beta+W]$ and $S[\delta+\beta+2W-1]$. Notice that S[$\delta+\beta+W$] has been reserved for the possible appearance of p_{l-1}, so the number of possible appearances of p_l (*w.r.t.* to p_{l-1} at $S[\delta+\beta+W-1]$) is $W-1$, unless p_{l-1} and p_l are the same, which is not a generic case in reality. Similarly, the number of possible appearances of p_l (*w.r.t.* to p_{l-1} at $S[\delta+\beta+W-2]$) is $W-2$. As a result, for all possible W appearances of p_{l-1} between $S[\delta+\beta+1]$ and $S[\delta+\beta+W]$, the sum of their possible matching p_l's occurrences is $\varphi_{l-1}=W+W-1+W-2\ldots+0=W(W+1)/2$.

So the maximal support for length-l pattern $P=p_1Qp_{l-1}p_l$ is $W\cdot sup(Q)\cdot \varphi_{l-1} = \sup(Q)\cdot(W^3+W^2)/2$, that is, $sup(P)$ • $\sup(Q)\cdot(W^3+W^2)/2\cdot$ □

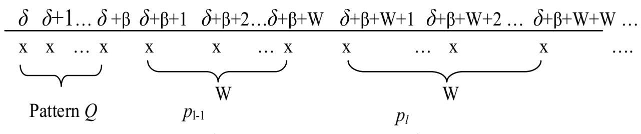

δ δ+1… δ+β δ+β+1 δ+β+2…δ+β+W δ+β+W+1 δ+β+W+2 … δ+β+W+W …
x x … x x x … x x … x x ….
W
W
Pattern Q
p_{l-1}
p_l

Figure 3. Pattern growing

LEMMA 1. Given a threshold ρ, we say that a length-l pattern P is frequent iff $sup(P)/L_l$ • ρ. If P's any length l-3 subpattern *Q's* frequency, $Freq(Q)$, is less than $\frac{L-(l-1)\cdot(\omega+1)}{L-(l-4)\cdot(\omega+1)}\cdot\rho$, where ω=(N+M)/2, then P cannot be a frequent pattern.

Proof:

Because $Freq(Q)=Sup(Q)/L_{l-3}$, we know $\frac{Sup(Q)}{L_{l-3}} \leq \frac{L-(l-1)\cdot(\omega+1)}{L-(l-4)\cdot(\omega+1)}\cdot\rho$, since $L_{l-3}=(L-(l-4)\cdot(\omega+1))\cdot W^{l-4}$,

We have $Sup(Q) < (L-(L-1)\cdot(\omega+1))\cdot W^{l-4}\cdot\rho$

Because Q is a length l-3 subpattern of P, according to Theorem 1, we know that $Sup(P) \leq \sup(Q)\cdot(W^3+W^2)/2$

So $Sup(P) < (L-(L-1)\cdot(\omega+1))\cdot W^{l-4}\cdot\rho\cdot(W^3+W^2)/2$

$$= (L-(L-1)\cdot(\omega+1))\cdot W^{l-1}\cdot\frac{W+1}{2W}\cdot\rho$$

$$\frac{Sup(P)}{(L-(L-1)\cdot(\omega+1))\cdot W^{l-1}} = \frac{Sup(P)}{L_l} < \rho\cdot\frac{W+1}{2W} \qquad (3)$$

Because the gap flexibility W •1, we know $(W+1)/2W \leq 1$, *i.e.*, $Sup(P)/L_l < \rho$. Therefore, P is not frequent. □

LEMMA 2: If the average span of the longest pattern in S is less than $(W-1)\cdot L/2W$, *i.e.*, about a half of a length-L hyper-sequence S, given a length-l pattern P, for any length l-3 subpattern of P, Q, if $Freq(Q)$ is less than $\frac{L-(l-1)\cdot(\omega+1)}{L-(l-4)\cdot(\omega+1)}\cdot\rho$, where $\omega=(N+M)/2$, then patterns, with P as their subpatterns, are not frequent.

Justification:
According to Lemma 1, we know that given the conditions in Lemm2, a length-l pattern P will not be frequent. Now assume pattern P is a subpattern of a length-$l+k$ pattern F. According to Eq. (2), we know $Sup(P) < \rho\cdot L_l\cdot(W+1)/2W$, for any length-$l+k$ pattern F, with P as its subpattern, the maximal support of F is less than W^k times of $Sup(P)$.

So $Sup(F) < W^k Sup(P) < \rho\cdot\frac{W+1}{2W}\cdot L_l\cdot W^k$. The frequency of F is

$$Freq(F) = \frac{Sup(F)}{L_{l+k}} < \rho\cdot\frac{W+1}{2W}\cdot\frac{L_l}{L_{l+k}}\cdot W^k$$

$Freq(F) < \rho\cdot\frac{W+1}{2W}\cdot\frac{(L-(l-1)\cdot(\omega+1))}{(L-(l+k-1)\cdot(\omega+1))}$, where $(l+k-1)\cdot(\omega+1)$ is the average span of the length-$l+k$ pattern. Given that the longest pattern is less than $(W-1)\cdot L/2W$, we have

$$Freq(F) < \rho\cdot\frac{W+1}{2W}\cdot\frac{(L-(l-1)\cdot(\omega+1))}{(L-L\cdot(W-1)/2W)}$$
$$= \rho\cdot\frac{W+1}{2W}\cdot\frac{2W}{W+1}\cdot\frac{L-(l-1)\cdot(\omega+1)}{L} < \rho$$

So pattern F is not frequent.

In reality, we may not know in apriori that whether the average span of the longest pattern in S is less than $(W-1)\cdot L/2W$ or not (since the longest patterns are yet to be found), so Lemma 2 does not seem to be useful in the mining process. Nevertheless, because we are dealing with a long hyper-sequence S, it is almost certain that the average span (even the maximal span) of the longest pattern in S is less than a half of $|S|$. For the *DNA* sequences we are using (in Section 7), the average span of the longest pattern is less than 10 percent of the sampled length-1000 sequence S. So we can safely assume that this prerequisite always holds.

5 Pattern Search with Gap Requirements

Consider a length-l pattern P with a gap flexibility W, an exhaustive search will start from the first pattern letter p_1 to find its first match in S. Denoting this matching location by x, the search process then starts from x to match p_2 within the range [$x+1$, $x+W$]. Such a process iteratively repeats until all possible locations starting from x have been checked, then it moves one step forward (x+1). The time complexity is $O(L\bullet W^{l-1})$, which is linear *w.r.t.* L, but exponential *w.r.t.* W and l. In the case that S consists of K element sequences, this complexity increases to $O(K\bullet L\bullet W^{l-1})$.

5.1 Gap Constrained Search (GCS)

In this subsection, we propose a Dynamic Programming [Bellman 57] oriented search mechanism, which is able to achieve a linear time complexity in gap constrained pattern search. The algorithm consists of three steps. Given a length-l pattern $P=p_1p_2\ldots p_l$, we first build a length L list for each of the element patterns p_l, denoted by O_{P_l}. We initiate the value of O_{P_l} to 0 before the search process (For easy understanding, we pictorially show a simple example in Figure 4 with P=AGTC and gap g_0^2).

1. *GCS* sequentially scans S from the left to right. For any current position x, if $S[x]$ matches the first element pattern p_1, set the value of $O_{P_1}[x]$ to 1.
2. At any location x, if $S[x]$ matches any element pattern p_j, j>1 (*i.e.*, excluding the first element pattern), we update the value of O_{P_j} to $O_{p_j} = \sum_{v=1}^{\max(1,x-W)} O_{p_{j-1}}[x-v]$.
   As shown in Figure 4, when x=3, S[3] matches p_2 (which is G), then the value of O_G[3] is updated to the sum of O_A[1] and O_A[2]. The above process indicates that for any matches of the element pattern letter, p_j j>1, at location x, we backtrack W steps to find the number of times p_j's last successive pattern letter p_{j-1} has ever appeared. If all element patterns (p_j, j>1) were able to iteratively regulate and update their lists O_{P_j} in such a way, then the value in $O_{P_j}[x]$ will indicate the number of times that pattern $p_1p_2\ldots p_j$ ever ends at position x.
3. We iteratively repeat the above process, until we finish the whole sequence S. Then we sum up all elements in O_{P_l}, which is the number of times $P=p_1p_2\ldots p_l$ appears in S. In addition, the values in $O_{P_l}[x]$ will indicate the number of times P ending at position x.

The time complexity of *GCS* consists of two parts: (1) scanning the whole sequence L, and (2) at each location x, comparing $S[x]$ with all element patterns, and backtracking W steps if necessary. Because each backtracking can be achieved through a sum operation, so the total time complexity is $O(KL(l$-1$))$, which is linear *w.r.t.* L, l, and K, and much more efficient than the exhaustive search $O(KLW^{l-1})$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
S	A	A	G	G	C	A	T	C	T	C	A	G	A	T	C	T	T	C
A	1	1	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0
G	0	0	2	2	0	0	0	0	0	0	0	1	0	0	0	0	0	0
T	0	0	0	0	0	0	2	0	0	0	0	0	0	1	0	0	0	0
C	0	0	0	0	0	0	0	2	0	2	0	0	0	0	1	0	0	0

Figure 4. Gap constrained pattern search (gap constraint g_0^2)

5.2 Map-based Support Prediction

Notice that each time we grow a length-l pattern P to a length l+1 pattern F, we will have to check F's frequency by searching its occurrences in S again. This reexamination mechanism costs a considerable amount of system runtime, as there are possibly millions of candidates. Nevertheless, if we can reuse P's occurrence information, we may be able to speed up the search process dramatically. For this purpose, after we find pattern P's occurrences, we generate a rear-map (RM) for P which records the number of times P appears in S and all its ending positions (this *RM* is actually the O_{P_l} list in the above section). As shown in Figure 4, if P ever ends at position x, the value of *RM*[x] will indicate the number of times P ends at x; otherwise, *RM*[x] equals to 0.

With the *RM* of pattern $P=p_lp_2\ldots p_l$, denoted by the $RM_{p_1\cdots p_l}[x]$, we may just search P's *RM*, instead of scanning the whole sequence S, to find the number of times F appearing in S. This can be achieved by a simple production and sum procedure. More specifically, for pattern $F=p_lp_2\ldots p_lp_{l+1}$, if we can build a head-map (*HP*) which records locations and times of the length-2 pattern p_lp_{l+1}'s starting information, where $HP_{p_lp_{l+1}}[x]$ indicates the number of times p_lp_{l+1} starts at position x, then, the support of pattern F is determined by Eq. (4)

$$Sup(p_1p_2\ldots p_{l+1}) = \sum\nolimits_{x=1}^{L} RM_{p_1\cdots p_l}[x]\cdot HP_{p_lp_{l+1}}[x] \quad (4)$$

As shown in Figure 5, when predicting the support of AGTC, we first find RM_{AGT} and HM_{TC}, the production and sum of the corresponding elements of these two lists will exactly equal to AGTC's support.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
S	A	A	A	G	C	G	T	C	T	C	A	G	G	T	C	T	T	C
RM$_{AGT}$	0	0	0	0	0	0	4	0	1	0	0	0	0	2	0	1	0	0
HM $_{TC}$	0	0	0	0	0	0	2	0	1	0	0	0	0	1	0	1	1	0
RM$_{AGTC}$	0	0	0	0	0	0	0	4	0	5	0	0	0	0	2	0	0	1

Figure 5. Map-based support prediction (gap constraint g_0^2)

6 Algorithm

The system framework of *MCPaS* is shown in Figure 6. *MCPaS* first generates all length-2 patterns, denoted by C_2. After the first step, *MCPaS* begins pattern growing and pruning. Assuming now at a certain step, we have generated a set of length-l candidates from length-l-1 patterns (on line 5 in Figure 6), then a length-l candidate's support can be easily predicted by plugging its length-l-1 patterns' *RM* and the corresponding length-2 patterns' *HM* into Eq. (4), without rescanning S. For all generated length-l candidates in C_l, we calculate a value $l'=l+3$ and a threshold,

$$\rho' = \frac{L-(l'-1)\cdot(\omega+1)}{L-(l'-4)\cdot(\omega+1)}\cdot\rho \quad (5)$$

All length-l candidates with their frequencies larger than ρ are forwarded to a frequent set F_l. If any length-l candidate's frequency is less than ρ', we mark it as "*suspicious*", which means that this pattern is unlikely to grow further, so we will keep an eye on it. Meanwhile, for any length-l candidate P in C_l, if any of P's length-l-2 subpattern is *suspicious*, we will remove P from the candidate set C_l (on lines 11 to 12 in Figure 6). According to Lemmas 1 and 2, if a pattern Q is suspicious, then any length-l patterns with Q as their length l-3 patterns are not going to be frequent. Therefore, if P's length-l-2 subpattern is *suspicious*, then any length l+1 (and beyond) patterns containing such subpatterns are not going to be frequent. So there is no need to put them into the candidate set C_l for growing. As a result, we may safely remove P from the candidate set C_l.

After *MCPaS* prunes out candidates from C_l, it builds *RM* for all remaining length-l patterns in C_l by rescanning S (on line 13 in Figure 6). *MCPaS* grows length-l+1 candidates by using all patterns in C_l (on line 5 in Figure 6). This can be achieved through the following two techniques: (1) trying all combinations by attaching any possible element pattern to the patterns in C_l, or (2) using the popular Apriori candidate generation procedure.

Input: (1) Hyper-sequence S and gap g_N^M, (2) # of element sequences K; (3) alphabet •; and (4) frequency threshold ρ
Output: Frequent pattern set

```
1.  $W \leftarrow M-N+1$
2.  Build length-2 pattern set $C_2$, build $BM$ and $HM$ maps
    for all patterns in $C_2$.
3.  $l \leftarrow 3$
4.  While ($C_{l-1}$ • $\phi$)
5.      $C_l \leftarrow PatternGen(C_{l-1})$;
6.      Predict support values for all candidates in $C_l$
        (Eq. (4))
7.      $l'=l+3$ AND calculate threshold $\rho'$
8.      For any pattern $y$ in $C_l$
9.          If $Freq(y)$ • $\rho$  Then  $F_l \leftarrow F_l \cup y$
10.         If $Freq(y) < \rho'$  Then  $y \leftarrow suspicious$
11.         If any length $l$-2 subset of $y$ is suspicious
12.                     Then  $C_l \leftarrow C_l \setminus y$
13.     Rescan $S$ and build $RM$ for all patterns in $C_l$
14.     $l \leftarrow l+1$;
15. Return ($F_3 \cup F_4 \ldots \cup F_{l-1}$)
```

Figure 6. MCPaS Algorithm

7 Experimental Results

The data used in our experiments are nucleotide DNA sequences downloaded from the National Center for Biotechnology Information website [NCBI], we choose four DNA sequences as our test bed (AX829168, AX829170, AX829174, and AX829178). When using multiple sequences to form a hyper-sequence, we truncate sequences into equal length ones. Because we use DNA sequences, the alphabet for all element sequences is Σ={A, C, T, G}. For comparisons, we implement the *MPPm* method in [Zhang et al. 05] in finding frequent patterns with gap constraints. This *MPPm* is the most relevant (and most recent as well) method we can find from all other peers.

Table 1 Candidate numbers scanned by different methods

Pattern	*Enumerate All*	MPP_m	$MCPaS_{Pruning}$	*MCPaS*
C_3	64	64	64	64
C_4	256	256	256	256
C_5	1024	1024	1024	1018
C_6	4096	4096	4088	3997
C_7	16384	16381	15535	11461
C_8	65536	54072	39728	7138
C_9	262144	19675	16108	1581
C_{10}	1048576	3459	2653	273
C_{11}	4194304	414	350	41
C_{12}	16777216	42	38	13
C_{13}	4^{13}	3	3	2

7.1 Pruning Efficiency Comparison

We provide in this subsection a pruning efficiency comparison with MPP_m by using a single *DNA* sequence AX829174 (by randomly sampling a *L*=1000 subsequence).

In Table 1, we report the experimental results (by the average results of 10 executions), where the first column means the candidate pattern set with different lengths. The second column indicates the number of candidates one has to evaluate, if enumerating all combinations. The third column means the number of candidates evaluated by MPP_m. Because *MCPaS* uses two approaches, map-based support prediction and Lemma 2, for pruning, we'd like to assess their efficiency separately. We first discard the map-based support prediction in the algorithm by replacing line 6 with line 13 in Figure 6. The results are denoted by $MCPaS_{Prunning}$. After that, we use both map-based support prediction and Lemma 2 for pruning, with results denoted by *MCPaS*.

When comparing $MCPaS_{Prunning}$ and MPP_m, we find that $MCPaS_{Prunning}$ has about 20% or fewer candidates than MPP_m. A further study on $MCPaS_{Prunning}$ and MMP_m reveals that they have opposite pruning mechanisms. In $MCPaS_{Prunning}$, patterns are growing and pruned orderly, which means that we generate length-*l* candidates, prune out unlikely ones, grow candidates and repeat the algorithm until the candidate set is empty. On the other hand, MPP_m uses reverse pruning mechanisms. It first determines the maximal length of the frequent pattern *n*, and based on this value, works out the minimal support values for different lengths of patterns. Not only the value *n* might be determined inaccurately (an inaccurate *n* will therefore reduce the pruning efficiency), even if *n* is perfectly determined, it will leave the selected threshold (for length *n*-1, *n*-2, …, 3) to be relatively small, because it has to consider the worst scenarios.

When combined with the map-based frequent prediction mechanism, *MCPaS* makes dramatic improvement in reducing the number of candidates in C_l. For example, the number of patterns needs to be scanned in C_8 is 7138, which is about 86.8% less than the number of patterns scanned by MMP_m.

7.2 Pattern Search Efficiency Comparison

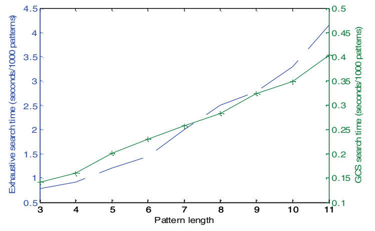


Figure 7. Pattern search efficiency (*W*=4, *L*=1000, AX829174)

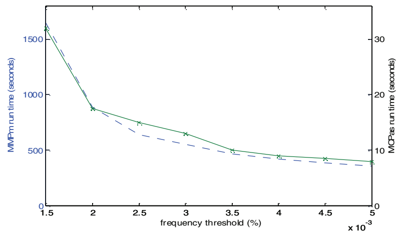


Figure 8. Pattern mining performance from single sequence

To assess the performances of the proposed *GCS* mechanism in comparison with exhaustive search, we sample a length *L*=1000 subsequence from AX829174, then use two search mechanisms and record their average search time (in seconds) for every 1000 patterns, and report the results in Figure 7 (the average results of 10 executions). In Figure 7, the *x*-axis denotes the length of the patterns, the dash line indicates the results of exhaustive search (corresponding to the *y*-axis on the left side of the figure), and the solid line with crosses represents the results from *GCS* (corresponding to the *y*-axis on the right side of the figure).

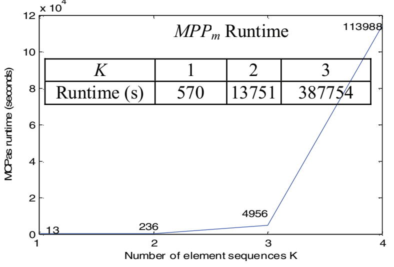
Comparing to exhaustive search, *GCS* is normally 5 to 10 times faster in searching (depending on the actual length of the patterns). Because exhaustive search's time complexity exponentially increases along with the pattern length and the gap flexibility, *GCS* can possibly achieve more improvements for longer patterns or larger gap constraints.

7.3 Pattern Mining Performance Comparison

To assess overall pattern mining performances, we first use a single sequence AX829174 with a fixed value *W*=4. We randomly sample a length *L*=1000 subsequence from AX829174, then use MPP_m and *MCPaS* to mine the results by specifying different threshold values ρ. We report the average runtime (from 10 executions) in Figure 8, where the *x*-axis means the value of ρ, the dash line indicates the results from MPP_m and the solid line with crosses denotes the results from *MCPaS* (corresponding to the *y*-axis on the left and right side of Fig. 8 respectively).

Both MPP_m and *MCPaS* nonlinearly respond to the threshold ρ with pretty similar shapes. This is because the value of ρ nonlinearly determines the number of candidates of the system. On average, *MCPaS* is about 40 times faster than MPP_m, with its improvement mainly comes from three aspects: (1) an ordinal pruning from Lemma 2; (2) map-based support prediction; and (3) *GCS* based search. *GCS* alone can enhance the search speed for about 8 times (comparing to exhaustive search), and the other two aspects will generally contribute a speed improvement of about 5 times.

In Figure 9, we report the results from the hyper-sequence consisting of one to up to four sequences (due to the intensive time consumption, we were only able to run the programs for only one time for 3 or 4 sequences). The *x*-axis in Figure 9 represents the number of element sequences, and the *y*-axis denotes the average runtime of *MCPaS*. Because MPP_m and *MCPaS* have huge runtime differences, we report *MPPm*'s runtime in a small table in Figure 9. Meanwhile, for a hyper-sequence with *K*=4 element sequences, MPP_m's runtime is too big, so we omit the value at this point.



K	1	2	3
Runtime (s)	570	13751	387754

Figure 9. Pattern mining performance from multiple sequences

The results in Figure 9 show that *MCPaS*'s runtime exponentially increases by the number of element sequences *K*. Although this sounds disappointing, to understand the challenge of our problem, let's assume that *S* merely consists of two *DNA* element sequences, then each element pattern p_i has $(|\Sigma|+1)^2-1=(4+1)^2-1=24$ possibilities (excluding the one consisting of wildcards only). So the number of length-5 candidate patterns is $24^5=7962624$, if no pruning techniques are involved. Although we can transform element sequences to form a hybrid sequence and apply MPP_m to solve the problem, because of the large alphabet size and the less effective reverse pruning technique, most of the length-5 candidates are going to be treated as frequent and used to grow next level candidates. *MCPaS* on the other hand, will start to prune candidates from length-4 candidates, and for length-5 patterns it will reduce about 80% of candidates (if *S* consists of 2 element sequences). The above observations make us believe that although *MCPaS* is nonlinear *w.r.t.* the number of element sequences *K*, when mining complex patterns from hyper-sequences, it is much more practical in reality.

8 Conclusions

We have studied in this paper the problem of mining complex patterns across multiple sequences with gap requirements, where patterns repetitively appear in multiple sequences and their matching appearances are flexibly confined by users' gap requirements. Because of the exponential candidate explosion, traditional Apriori-based solutions are technically infeasible to solve the problem. We have proposed *MCPas* with three unique features to fulfill the task: (1) an Apriori-like mining framework which allows pattern generation and growing to be conducted step by step; (2) map-based support predication to predict candidates' frequency without rescanning; and (3) a gap constrained linear-time pattern search. Experimental comparisons have shown that each of the above techniques has made a contribution, and the overall performances of *MCPas* have been about 40 times faster than its other peers.

References

[Altschul 90] S. Altschul, W. Gish, W. Miller, E. Myers, & D. Lipman, Basic local alignment search tool. *Journal of Molecular Biology*, 215:403-410, 1990.

[Bellman 57] R. E. Bellman. *Dynamic Programming*, Princeton University Press, 1957.

[Belkum et al. 97] A. van Belkum, S. Scherer amd W. van Leeuwen, D. Willemse, L. van Alphen, & H. Verbrugh. Variable number of tandem repeats in clinical strains of haemophilus influenzae. *Infection and Immunity*, 65(12):5017-5027, 1997.

[Chen et al. 05] G. Chen, X. Wu, & X. Zhu, Sequential pattern mining in multiple data streams, *Proc. of ICDM*, TX, 2005.

[Das et al. 98] G. Das, K. Lin, H. Mannila, G. Renganathan, & P. Smyth, Rule discovery from time series, *Proc. of KDD*, 1998.

[Herzel et al. 99] H. Herzel, O. Weiss, & E. Trifonov, 10-11 BP periodicities in complete genomes reflect protein structure and DNA folding, *Bioinformatics*, 15(3):187-193, 1999.

[Jonassen 97] I. Jonassen, Efficient discovery of conserved patterns using a pattern graph, *Computer Applications in the Biosciences*, 13:509-522, 1997.

[Mannila et al. 97] H. Mannila, H. Toivonen, and A.I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining & Knowledge Discovery*, 1(3), 1997.

[Méger & Rigotti 04] N. Méger, C. Rigotti: Constraint-based mining of episode rules and optimal window. *Proc. of PKDD*, 2004.

[Murray et al. 02] K. Murray, D. Gorse, & J. Thornton, Wavelet transforms for the characterization and detection of repeating motifs, *Journal of Molecular Biology*, 316:341-363, 2002.

[NCBI] NCBI: http://www.ncbi.nlm.nih.gov

[Oats & Cohen 96] T. Oates & P. Cohen, Searching for structure in multiple streams of data, *Proc. of ICML*, 1996.

[Pei et al. 01] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, & M. Hsu. *PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth*, *Proc. of ICDE*, 2001.

[Pino et al. 01] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, & U. Dayal, Multi-dimensional sequential pattern mining, *Proc. of CIKM*, 2001.

[Rigoutsos & Floratos 98] I. Rigoutsos & A. Floratos. Combinatorial pattern discovery in biological sequences: the teiresias algorithm. *Bioinformatics*, 14(1), 1998.

[Srikant & Agrawal 96] R. Srikant & R. Agrawal, Mining Sequential Patterns: Generalizations and Performance Improvements, *Proc. of the International Conf. on Extending DB Tech.*, 1996.

[Yang et al. 00] J. Yang, W. Wang, & P. Yu, Mining asynchronous periodic patterns in time series data, *Proc. of KDD*, MA, 2000.

[Zaki 98] M. Zaki, Efficient enumeration of frequent sequences, *Proc. of CIKM*, November 2-7, 1998, Bethesda, Maryland.

[Zhang et al. 05] M. Zhang, B. Kao, D. Cheung, & K. Yip, Mining periodic patterns with gap requirement from sequences, *Proc. of ACM SIGMOD*, Baltimore Maryland, 2005.