# Frequent Pattern Mining with Non-overlapping Inversions

Da-Jung Cho, Yo-Sub Han(✉), and Hwee Kim

Department of Computer Science, Yonsei University, 50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
{dajung,emmous,kimhwee}@cs.yonsei.ac.kr

**Abstract.** Frequent pattern mining is widely used in bioinformatics since frequent patterns in bio sequences often correspond to residues conserved during evolution. In bio sequence analysis, non-overlapping inversions are well-studied because of their practical properties for local sequence comparisons. We consider the problem of finding frequent patterns in a bio sequence with respect to non-overlapping inversions, and design efficient algorithms.

**Keywords:** String processing algorithms · Frequent pattern mining · Non-overlapping inversions

## 1 Introduction

Agrawal et al. [1] studied the frequent pattern mining for finding associations among market products and increasing profit. For example, frequent patterns in customer behavior are useful for setting affordable product price, promotion and store layout. They investigated the problem of finding meaningful associations over market transactions. Frequent patterns are also useful in other domains including sequential data, bio sequences or strings [10–12,14,18,20].

In bioinformatics, frequent motifs in DNA or protein sequences often correspond to residues conserved during evolution due to an important structural or functional role [18]. Note that traditional pattern mining algorithms are not suitable for bio sequences, since they cope with a large number of items and short sequence lengths [11]. Wang et al. [18] first proposed an algorithm that finds sequential patterns on bio sequences. Recently, Liao and Chen [12] designed an algorithm for the problem with gaps—regions not conserved in evolution.

From a biological aspect, an inversion—breakage and rearrangement within itself—is one of the most important operations since such an event produces new gene sequences from an original gene sequence and sometimes causes a disease [13]. Schöniger and Waterman [15] introduced a simplification hypothesis that all regions involved in the inversions do not overlap. This hypothesis—non-overlapping inversions—is realistic for local DNA comparisons on relatively closed sequences [17]. On the string with non-overlapping inversions, Chen et al. [4] designed an $O(n^4)$ algorithm to solve the alignment with non-overlapping inversions, which was improved to $O(n^3)$ by Vellozo et al. [17]. Amir


A.-H. Dediu et al. (Eds.): LATA 2015, LNCS 8977, pp. 121–132, 2015.
DOI: 10.1007/978-3-319-15579-1_9

and Porat [2] proposed an $O(n^2)$ approximation algorithm for the problem, and Cho et al. [6] proposed an $O(n^3)$ algorithm for the modified problem where inversions occur to both strings. For the pattern matching, Cantone et al. [3] proposed $O(nm)$ algorithm to solve the pattern matching with non-overlapping inversions. On formal language theory, researchers [5,8] have studied properties and decision problems of formal languages considering inversions. This leads us to consider frequent pattern mining problem on a string with non-overlapping inversions.

Due to the irregularity of gene evolution, rearrangements—for instance, non-overlapping inversions—may occur in conserved regions $P$. Suppose that a sequence $S_{i+1}$ is obtained from a sequence $S_i$ that has conserved regions, and non-overlapping inversions occur during the evolution from $S_i$ to $S_{i+1}$ (See Fig. 1 for an illustrative example of this phenomenon.). We search for a conserved region $P$ in $S_{i+1}$, which now may have been modified by non-overlapping inversions from $S_i$. Note that often we do not have all evolution sequences of a gene. For instance, here $S_{i+1}$ is a mere sequence that we have and, thus, we do not know where exactly non-overlapping inversions occur in $S_{i+1}$. This makes the problem of finding similar or same pattern occurrences in $S_i$ challenging when we have only $S_{i+1}$ and the fact that $S_{i+1}$ is generated from $S_i$ by some non-overlapping inversions.

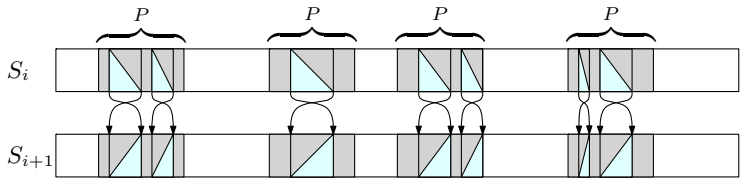


**Fig. 1.** Let $S_i$ be a gene sequence of the $i$th generation and $S_{i+1}$ be a gene sequence of the $i+1$th generation. During the evolution from $S_i$ to $S_{i+1}$, non-overlapping inversions flip subsequences of the conserved regions $P$.

We formulate our problem as a frequent pattern mining problem on a string: Given a text $T$ of length $n$ over an alphabet $\Sigma$, a pattern length $m$ and a pattern occurrence threshold $r$, our goal is to compute the set of all patterns $P$ of length $m$ that occur in $T$ at least $r$ times when we allow non-overlapping inversions on $P$. Note that $P$ may not be a substring or a subsequence of $T$, which is different from other frequent pattern mining problems in the literature. We first compute a set of all possible substrings $T_i$ of length $m$ and construct digraphs $G_i$ representing all strings that can be generated by non-overlapping inversions on $T_i$. Next, we overlay all such $G_i$'s and obtain a weighted multidigraph $G$. Then, we find all paths in $G$ with the bottleneck lower bound $r$—each path represents $P$ and the bottleneck is the number of occurrences of $P$. We show that we can find all patterns in $O(nm^2 + Cm)$ time using $O(m)$ space, where $C$ is the number of matching patterns. If we want to store all matching patterns instead of reporting

them, then we can construct a DFA that recognizes the set of all such patterns in $O(nm^2)$ time using $O(m)$ space.

## 2 Preliminaries

Let $A[a_1][a_2]\cdots[a_n]$ be an $n$-dimensional array, where the size of each dimension is $a_i$ for $1 \leq i \leq n$. Let $A[i_1][i_2]\cdots[i_n]$ be the element of $A$ with indices $(i_1, i_2, \ldots, i_n)$. Given a finite set $\Sigma$ of characters and a string $w$ over $\Sigma$, we use $|w|$ to denote the length of $w$ and $w[i]$ to denote the symbol of $w$ at position $i$. We use $w[i:j]$ to denote the substring $w[i]\cdots w[j]$, where $0 < i \leq j$.

For a finite set $\Sigma$ of characters, $\Sigma^*$ denotes the set of all strings over $\Sigma$. A language over $\Sigma$ is any subset of $\Sigma^*$. The symbol $\emptyset$ denotes the empty language and the symbol $\lambda$ denotes the null string. A finite-state automaton (FA) $A$ is specified by $A = (Q, \Sigma, \delta, s, F)$, where $Q$ is a set of states, $\Sigma$ is an alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, $s \in Q$ is the start state and $F \subseteq Q$ is a set of final states. For a transition $\delta(p, \sigma) = q$, we say that $p$ has an *out-transition* and $q$ has an *in-transition*. Moreover, we call $q$ a *target state* of $p$. A string $w$ is accepted by $A$ if there is a labeled path from $s$ to a final state in $F$ such that the path spells out $w$. The language $L(A)$ of an FA $A$ is the set of all strings accepted by $A$. If $|\{\delta(p, \sigma)\}| = 1$ for all $p \in Q$ and $\sigma \in \Sigma$, we say that $A$ is a deterministic finite-state automaton (DFA); otherwise, $A$ is a nondeterministic finite-state automaton (NFA). For more knowledge in automata theory, the reader may refer to textbooks [16,19].

We consider a biological operation *inversion* $\theta$ and denote by $\theta(w)$ the reverse of a string $w$. We define an inversion operation $\theta_{(i,j)}$ for a given range $(i, j)$ to be $\theta_{(i,j)}(w) = \theta(w[i : j])$. When the context is clear, we denote $\theta_{(i,j)}$ as $(i, j)$. We say that the length of $(i, j)$ is $j - i + 1$. We define a sequence $\Theta = ((p_1, q_1), (p_2, q_2), \ldots, (p_k, q_k))$ of inversions for a string $w$ to be *non-overlapping* (NOI-sequence for short) if it satisfies the following conditions: For $1 \leq i \leq k$, $p_1 \geq 1$, $q_k \leq |w|$, $p_i \leq q_i$ and $p_{i+1} \geq q_i + 1$ for $1 \leq i \leq k - 1$. For the sake of easier explanation of our algorithms, for any given index $i$, we assume that there always exists an inversion whose range covers $i$; namely, $p_1 = 1$, $q_k = |w|$ and $p_{i+1} = q_i + 1$, since a non-inversed range $(i, j)$ can be represented by a sequence $((i, i), (i{+}1, i{+}1), \ldots, (j, j))$ of inversions. Now, in summary, given an NOI-sequence $\Theta = ((p_1, q_1), (p_2, q_2), \ldots, (p_k, q_k))$ and a string $w$, $\Theta(w) = \theta(w[p_1 : q_1])\theta(w[p_2 : q_2])\cdots\theta(w[p_k : q_k])$.

An undirected graph $G = (V, E)$ consists of a finite nonempty set $V$ of nodes and a set $E$ of unordered pairs of distinct nodes of $V$. Each pair $e = \{u, v\}$ of nodes in $E$ is an edge of $G$ and $e$ is said to join $u$ and $v$. A directed graph or digraph $D$ consists of a finite nonempty set $V$ of nodes and a set $E$ of ordered pairs of nodes. For an edge $e = (u, v)$ of a digraph, we say that $e$ is from node $u$ to node $v$. A multidigraph is a digraph where more than one edge can join two nodes. The reader may refer to Harary [7] for more details in graph theory.

Given two strings $X$ and $Y$ of the same length, we say that $X$ and $Y$ have an *alignment with non-overlapping inversions* (NOI-alignment for short) if there

exists an NOI sequence $\Theta$ such that $\Theta(Y) = X$. Given a text $T$ of length $n$ and a pattern $P$ of length $m$, the *NOI-occurrence* $Occ(T, P)$ of $T$ and $P$ is the number of indices $i$ where $T_i = T[i : i+m-1]$ has an NOI-alignment with $P$.

**Definition 1 (Frequent Pattern Mining with Non-overlapping Inversions).** *Given a text $T$ of length $n$ over $\Sigma$, a pattern length $m$ and a minimum number $r$ of pattern occurrences, find all pairs $(P \in \Sigma^m, Occ(T, P))$ where $Occ(T, P) \geq r$.*

## 3 The Algorithm

Given a text $T$, our algorithm starts from inspecting all substrings $T_i$ of $T$. We first compute a set of all NOI-alignment strings for $T_i$ and construct a digraph $G_i$ that represents the set. Then we overlay all $G_i$'s and construct a weighted digraph $G$, and find all frequent patterns $P$ from $G$, where $Occ(T, P) \geq r$. We construct inversion fragment tables for all substrings $T_i$.

**Definition 2.** *Given a text $T$, an index $i$ and a pattern length $m$, the* inversion fragment table *(*IFT *for short) is defined as follows:*

$$F_i[j][k] = \begin{cases} ((k, j), T_i[k]) & \text{if } k \leq j, \\ ((j, k), T_i[k]) & \text{otherwise} \end{cases}$$

*for $1 \leq j, k \leq m$.*

We call all elements in $F_i[j][k]$ *inversion fragments* (IFs for short) of $T_i$. For an IF $\mathbb{F} = ((p, q), \sigma)$, we say that $\mathbb{F}$ *yields* the character $\sigma$. For a sequence of IFs $\mathbb{F}_1, \ldots, \mathbb{F}_l$, where $\mathbb{F}_i$ yields $\sigma_i$, we say that the sequence yields a string $\sigma_1 \cdots \sigma_l$. Fig. 2 shows an example of an IFT.

| $F_1$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | $((1,1),A)$ | $((1,2),A)$ | $((1,3),A)$ | $((1,4),A)$ |
| 2 | $((1,2),G)$ | $((2,2),G)$ | $((2,3),G)$ | $((2,4),G)$ |
| 3 | $((1,3),C)$ | $((2,3),C)$ | $((3,3),C)$ | $((3,4),C)$ |
| 4 | $((1,4),T)$ | $((2,4),T)$ | $((3,4),T)$ | $((4,4),T)$ |

**Fig. 2.** IFT $F_1$ for $T = AGCTA$ and $m = 4$. Shaded cells denote IFs for $\theta_{(2,4)}(T_1)$.

IFs become useful for computing a substring created by an inversion because of the following property of the inversion operation:

**Observation 3.** *For a text $T$, an index $i$ and its IFT $F_i$, a sequence $F_i[j][k]$, $F_i[j+1][k-1], \ldots, F_i[k-1][j+1], F_i[k][j]$ of IFs yields $\theta_{(j,k)}(T_i)$.*

From Observation 3, we know that if we apply $\theta_{(\min(j,k),\max(j,k))}$ to $T_i$, then $\sigma$ yielded by $F_i[j][k]$ becomes the $j$th character of the result string. Using Observation 3, we construct a digraph $G_i$ for $T$ and $i$ by Algorithm 1, which we call an *inversion graph*. In an inversion graph, each node $(j, \sigma)$ represents that there exists an NOI-alignment between $P$ and $T_i$ where $P[j] = \sigma$. Each edge $((j, \sigma_1), (j{+}1, \sigma_2), \sigma_0, f = c \text{ or } v)$ represents that if $P[j{-}1] = \sigma_0$ and $P[j] = \sigma_1$, then we can set $P[j{+}1] = \sigma_2$ to ensure that $P$ and $T_i$ have an NOI-alignment. The last element $f$ is the flag that indicates whether $P[j]$ and $P[j{+}1]$ are from a single inversion on the text ($v$) or from two adjacent inversions ($c$). Fig. 3 shows an example of an inversion graph.

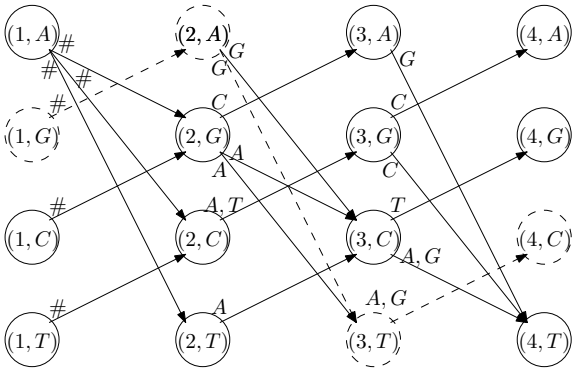


**Fig. 3.** An inversion graph $G_1$ for $T = AGCTA$ (flags of the edges are omitted). The dashed path represents $P = GATC$, which has an NOI-alignment with $T_1 = AGCT$.

We can retrieve all strings that have an NOI-alignment with $T_i$ from $G_i$.

**Theorem 4.** *Given a text $T$, an index $i$, a pattern length $m$ and an inversion graph $G_i$, a string $P$ has an NOI-alignment with $T_i$ if and only if*

1. $((1, P[1]), (2, P[2]), \#, f) \in E_i$ *and*
2. $((j, P[j]), (j{+}1, P[j{+}1]), P[j{-}1], f) \in E_i$ *for* $2 \leq j \leq m - 1$.

In the literature, people often assume that the alphabet size is constant, since $\Sigma$ is a finite set. Then, it is straightforward to verify that Algorithm 1 runs in $O(m^3)$ time. Since the size of $F_i$ is $O(m^2)$ and the size of $E_i$ is $O(m)$, Algorithm 1 requires $O(m^2)$ space. We improve the runtime of Algorithm 1 by relying on the following property of IFTs.

**Observation 5.** *For $1 \leq i < n - m$ and $2 \leq j, k \leq m$,*

1. *If $F_i[j][k] = ((j, k), \sigma)$, then $F_{i+1}[j{-}1][k{-}1] = ((j{-}1, k{-}1), \sigma)$.*
2. *If $F_i[j][k] = ((k, j), \sigma)$, then $F_{i+1}[j{-}1][k{-}1] = ((k{-}1, j{-}1), \sigma)$.*

**Algorithm 1.** ConstructInversionGraph

**Input**: Text $T$ over $\Sigma$ of size $t$, index $i$ and a pattern length $m$
**Output**: Digraph $G_i = (V, E_i)$, where $V = \{1, 2, \ldots, m\} \times \Sigma$ and $E_i \subset V \times V \times (\Sigma \cup \{\#\}) \times \{v, c\}$

```
1  construct F_i
2  V ← {1, 2, ..., m} × Σ
3  for j ← 1 to m − 1 do
       /* check the case that an inversion ends at index j */
4      for k ← 1 to j do
5          let σ1 be yielded by F_i[j][k]
           /* find all first characters of inversions starting from
              index j + 1 and record the edge */
6          for l ← j + 1 to m do
7              let σ2 be yielded by F_i[j+1][l]
8              if j = 1 then
9                  add ((j, σ1), (j+1, σ2), #, c) to E_i
10             else
11                 for each σ0 where ((j−1, σ0), (j, σ1), σ', f) ∈ E_i do
12                     add ((j, σ1), (j+1, σ2), σ0, c) to E_i
       /* check the case that index j is within the range of an
          inversion */
13     for k ← 2 to m do
           /* find the next character in the inversion and record the
              edge */
14         let σ1 be yielded by F_i[j][k] and σ2 be yielded by F_i[j+1][k−1]
15         if j = 1 then
16             add ((j, σ1), (j+1, σ2), #, v) to E_i
17         else if k ≠ m then
18             add ((j, σ1), (j+1, σ2), σ0, v) to E_i, where F_i[j−1][k+1] yields σ0
19         if j < k and j ≠ 1 then
20             for each σ0 where ((j−1, σ0), (j, σ1), σ', c) ∈ E_i do
21                 add ((j, σ1), (j+1, σ2), σ0, v) to E_i
22 return (V, E_i)
```

From Observation 5, we know that $G_i$ and $G_{i+1}$, constructed from $F_i$ and $F_{i+1}$ respectively, have common edges (See Fig. 4.). We make use of these edges to reduce the construction time of $G_{i+1}$ by adding a new label $(k, k')$ for each edge $((j, \sigma_1), (j+1, \sigma_2), \sigma_0, f) \in E_i$, representing that the edge is from IFs $F_i[j][k]$ and $F_i[j+1][k']$. Moreover, in Algorithm 1, $F_i$ is only used to compute characters yielded by IFs. Since $F_i[j][k]$ always yields $T_i[k]$, we do not need to construct $F_i$ in the algorithm. By modifying Algorithm 1, we obtain a new algorithm with improved time and space complexity. Algorithm 2 preserves all edges in $E_{i-1}$ that are made from IFs in $F_i$ and runs Algorithm 1 only for edges that are not

in $E_{i-1}$ but in $E_i$. While adding new edges to $E_i$, Algorithm 2 does not use $F_i$. Thus, Algorithm 2 constructs the same $G_i$ as Algorithm 1 except for additional labels.

**Algorithm 2.** ConstructFastInversionGraph

**Input**: Text $T$ over $\Sigma$ of size $t$, index $i$, pattern length $m$ and inversion graph $G_{i-1} = (V, E_{i-1})$
**Output**: Digraph $G_i = (V, E_i)$, where $V = \{1, 2, \ldots, m\} \times \Sigma$ and $E_i \subset V \times V \times (\Sigma \cup \{\#\}) \times \{v, c\} \times \{1, 2, \ldots, m\}^2$

`/* retrieve common edges from` $E_{i-1}$ `*/`
**for each** $((j, \sigma_1), (j{+}1, \sigma_2), \sigma_0, f, k, k') \in E_{i-1}$ **do**
  **if** $2 \le j \le m-1$ *and* $2 \le k, k'$ **then**
    add $((j{-}1, \sigma_1), (j, \sigma_2), \sigma_0, f, k{-}1, k'{-}1)$ to $E_i$
**for** $j \leftarrow 1$ **to** $m-1$ **do**
  `/* add only new edges for` $j$ `using a part of Algorithm 1 */`
  **for** $k \leftarrow 1$ **to** $j$ **do**
    $\sigma_1 \leftarrow T_i[k], \sigma_2 \leftarrow T_i[m]$
    **if** $j = 1$ **then** add $((1, \sigma_1), (2, \sigma_2), \#, c, k, m)$ to $E_i$ **else**
      **for each** $\sigma_0$ *where* $((j{-}1, \sigma_0), (j, \sigma_1), \sigma', f, k', k) \in E_i$ **do**
        add $((j, \sigma_1), (j{+}1, \sigma_2), \sigma_0, c, k, m)$ to $E_i$
  `/* add edges for` $j = m-1$ `using a part of Algorithm 1 */`
  **if** $j = m-1$ **then**
    **for** $k \leftarrow 2$ **to** $m-1$ **do**
      $\sigma_1 \leftarrow T_i[k], \sigma_2 \leftarrow T_i[k{-}1]$
      add $((m{-}1, \sigma_1), (m, \sigma_2), T_i[k{+}1], v, k, k{-}1)$ to $E_i$
  $\sigma_1 \leftarrow T_i[m{-}1], \sigma_2 \leftarrow T_i[m]$
  **for each** $\sigma_0$ *where* $((m{-}2, \sigma_0), (j, \sigma_1), \sigma', c, k', k) \in E_i$ **do**
    add $((j, \sigma_1), (j{+}1, \sigma_2), \sigma_0, v, k, k{-}1)$ to $E_i$
**return** $(V, E_i)$

| $F_1$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | $((1,1),A)$ | $((1,2),A)$ | $((1,3),A)$ | $((1,4),A)$ |
| 2 | $((1,2),G)$ | $((2,2),G)$ | $((2,3),G)$ | $((2,4),G)$ |
| 3 | $((1,3),C)$ | $((2,3),C)$ | $((3,3),C)$ | $((3,4),C)$ |
| 4 | $((1,4),T)$ | $((2,4),T)$ | $((3,4),T)$ | $((4,4),T)$ |

| $F_2$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | $((1,1),G)$ | $((1,2),G)$ | $((1,3),G)$ | $((1,4),G)$ |
| 2 | $((1,2),C)$ | $((2,2),C)$ | $((2,3),C)$ | $((2,4),C)$ |
| 3 | $((1,3),T)$ | $((2,3),T)$ | $((3,3),T)$ | $((3,4),T)$ |
| 4 | $((1,4),A)$ | $((2,4),A)$ | $((3,4),A)$ | $((4,4),A)$ |

**Fig. 4.** Comparison of $F_1$ and $F_2$ for $T = AGCTA$. Shaded cells denote IFs from $F_1$ and $F_2$ that satisfy the properties of Observation 5.

Since moving common edges from $E_{i-1}$ to $E_i$ takes $O(m)$ time and adding new edges takes $O(m^2)$ time, Algorithm 2 runs in $O(m^2)$ time using $O(m)$ space. Note that $G_i$ represents all possible strings that have NOI-alignments with $T_i$. We overlay all $G_i$'s to construct an *accumulated inversion graph* $G$ for $T$ by Algorithm 3 (See Fig. 5 for an example.). Note that Algorithm 3 requires $O(nm^2)$ time and $O(m)$ space.

**Algorithm 3.** ConstructAccumulatedInversionGraph

**Input**: Text $T$ of length $n$ over $\Sigma$ of size $t$ and a pattern length $m$
**Output**: Weighted multidigraph $G = (V, E)$, where $V = \{1, 2, \ldots, m\} \times \Sigma$ and $E \subset V \times V \times (\Sigma \cup \{\#\}) \times \{1, 2, \ldots, n - m + 1\}$

1 $V \leftarrow \{1, 2, \ldots, m\} \times \Sigma$
2 **for each** $(i, \sigma_1), (i{+}1, \sigma_2) \in V$, $\sigma_0 \in \Sigma$ **do**
3 add $((i, \sigma_1), (i{+}1, \sigma_2), \sigma_0, 0)$ to $E$.
4 **for** $i \leftarrow 1$ **to** $n - m + 1$ **do**
`/* modify ConstructInversionGraph not to use` $F_1$ `*/`
5 **if** $i = 1$ **then** ConstructInversionGraph$(T, i, m)$ **else** ConstructFastInversionGraph$(T, i, m, G_{i-1})$ **for each** $(v_1, v_2, \sigma, f, k, k') \in E_i$ **do**
6 change $(v_1, v_2, \sigma, g) \in E$ to $(v_1, v_2, \sigma, g + 1)$
7 **return** $(V, E)$

For an edge $((i, \sigma_i), (i{+}1, \sigma_{i+1}), \sigma_{i-1}, g_i) \in E$, we call $g_i$ the *weight* of the edge and $\sigma$ the *preceding symbol* of the edge. We also say that the edge is from index $i$ to index $i{+}1$. For a path $((1, \sigma_1), (2, \sigma_2), \#, g_1), ((2, \sigma_2), (3, \sigma_3), \sigma_1, g_2), \ldots, ((m{-}1, \sigma_{m-1}), (m, \sigma_m), \sigma_{m-2}, g_m)$, we call $\min(g_1, g_2, \ldots, g_m)$ the *minimum weight* of the path. From the construction of $G$, we have the following statement:

**Lemma 6.** *For a text $T$ of length $n$ and a pattern length $m$, let $P \in \Sigma^m$ be a pattern such that $((1, P[1]), (2, P[2]), \#, g_1) \in E$ and $((j, P[j]), (j{+}1, P[j{+}1]), P[j{-}1], g_j) \in E$ for $2 \le j \le m - 1$. Then $Occ(T, P) = \min(g_j)$ for $1 \le j \le m - 1$.*

Now, our goal is to find all paths from index 1 to index $m$, where the minimum weight of each path is greater than or equal to $r$. We reduce the resulting accumulated inversion graph so that any path from index 1 to index $m$ in the graph satisfies the condition based on a modified Kruskal's Algorithm [9]. We sort edges by ascending order with respect to weights as in Kruskal's Algorithm. Then we repeatedly remove an edge with the minimum weight until all remaining edges have weights greater than or equal to $r$. Once we remove an edge $e$, we then remove all adjacent edges of $e$ that cannot be in a path anymore because of the removal of $e$. Algorithm 4 removes edges from $G$ to return a graph $G'$, where any path from index 1 to index $m$ represents a pattern $P$ such that $Occ(T, P) \ge r$. Fig. 6 is an example of $G'$ for $T = AGCTAGCTAG$ and $r = 3$.

We prove that any path from index 1 to index $m$ in the resulting graph represents a pattern $P$ such that $Occ(T, P) \ge r$.

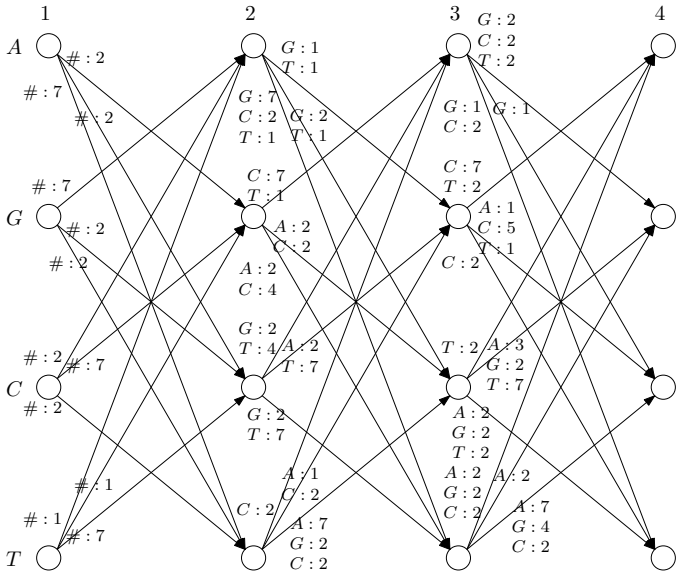


**Fig. 5.** An accumulated inversion graph $G$ for $T = AGCTAGCTAG$

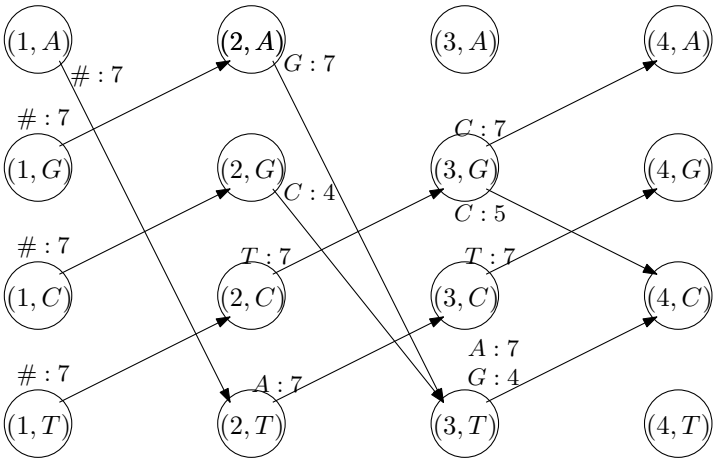


**Fig. 6.** A reduced accumulated inversion graph $G'$ for $T = AGCTAGCTAG$ and $r = 3$

**Theorem 7.** *Suppose Algorithm 4 returns* $G' = (V, E')$. *Let* $P \in \Sigma^m$ *be a pattern such that* $((1, P[1]), (2, P[2]), \#, g_1) \in E'$ *and* $((j, P[j]), (j+1, P[j+1]), P[j-1], g_j) \in E$ *for* $2 \leq j \leq m-1$. *Then* $Occ(T, P) \geq r$.

Next, we analyze the time and space complexity of the algorithm.

**Lemma 8.** *Algorithm 4 runs in* $O(nm^2)$ *time using* $O(m)$ *space.*

**Algorithm 4.** ReduceAccumulatedInversionGraph

**Input**: Text $T$ of length $n$ over $\Sigma$ of size $t$, a pattern length $m$ and a pattern occurrence threshold $r$
**Output**: Weighted multidigraph $G' = (V, E')$, where $V = \{1, 2, \ldots, m\} \times \Sigma$ and $E \subset V \times V \times (\Sigma \cup \{\#\}) \times \{1, 2, \ldots, n - m + 1\}$

```
1  ConstructAccumulatedInversionGraph(T, m)
2  E' ← E
3  sort E' by ascending order with respect to weights
4  while there exists an edge in E' with weight less than r do
5      e ← ((i, σ1), (i+1, σ2), σ0, g) be the edge with minimum weight in E'
6      R ← ∅                                          // set of edges to remove
7      if e is the only edge from index i to index i + 1 in E' then return (V, ∅)
       else add e to R while R ≠ ∅ do
8          for each e' ← ((j, σ'1), (j+1, σ'2), σ'0, g') ∈ R do
9              if e' is the only edge from node (j, σ'1) to node (j+1, σ'2) in E' then
10                 add all edges from node (j+1, σ'2) to index j+2 with preceding
                   symbol σ'1 in E' to R
11                 if there is no edge from node (j, σ'1) to index j+1 in E' then
12                     add all edges from index j−1 to node (j, σ'1) in E' to R
13             if e' is the only edge from node (j, σ'1) to index j+1 with preceding
               symbol σ'0 in E' then
14                 add all edges from node (j−1, σ'0) to node (j, σ'1) in E' to R
15             remove e' from E' and R
16 return (V, E')
```

It requires at least $O(Cm)$ to report all patterns with the occurrence greater than or equal to $r$, where $C$ is the number of such patterns. From $G'$, we can find all such patterns by simple depth-first search in $O(Cm)$. On the other hand, if we want to identify all matching patterns instead of reporting them, then we can convert $G'$ to a DFA $A'$, where $L(A')$ is the set of all such patterns. For a weighted multidigraph $G' = (V, E')$, where $V = \{1, 2, \ldots, m\} \times \Sigma$ and $E \subset V \times V \times (\Sigma \cup \{\#\}) \times \{1, 2, \ldots, n-m+1\}$, we construct a DFA $A' = (Q, \Sigma, \delta, s, F)$ by the following procedure:

1. $Q = s \cup \{\#\} \times \Sigma \times \{1\} \cup \Sigma \times \Sigma \times \{2, \ldots, m\}$.
2. $F = \Sigma \times \Sigma \times \{m\}$.
3. $\delta(s, \sigma) = (\#, \sigma, 1)$ for all $((1, \sigma), (2, \sigma'), \#, g) \in E'$, where $\sigma, \sigma' \in \Sigma$ and $g \geq 0$.
4. $\delta((\sigma_0, \sigma_1, i), \sigma_2) = (\sigma_1, \sigma_2, i+1)$ for all $e' \leftarrow ((i, \sigma_1), (i+1, \sigma_2), \sigma_0, g) \in E'$, where $\sigma_0, \sigma_1, \sigma_2 \in \Sigma$, $g \geq 0$ and $1 \leq i \leq m - 1$.

From the construction of the transition function, it is straightforward that $A'$ is a DFA and $L(A')$ is equal to the set of all patterns with the occurrence greater than or equal to $r$. The construction of $A'$ requires $O(m)$ time using $O(m)$ space. Then, we establish the following theorem.

**Theorem 9.** *Given a text $T$ of length $n$ over an alphabet $\Sigma$, a pattern length $m$ and a pattern occurrence threshold $r$, we can solve frequent pattern mining with non-overlapping inversions in $O(nm^2+Cm)$ time using $O(m)$ space, where $C$ is the number of patterns to find. Moreover, we can construct a DFA that recognizes the set of all patterns to find in $O(nm^2)$ time using $O(m)$ space.*

## 4 Conclusions

We have considered non-overlapping inversions in frequent pattern mining. We have proposed a graph-based algorithm that finds all patterns $P$ where $Occ(T, P) \geq r$ in $O(nm^2 + Cm)$ time using $O(m)$ space, where $m$ is the desired pattern length, $n$ is the size of an input text $T$, $r$ is the pattern occurrence threshold and $C$ is the number of patterns to find. Moreover, we have proposed an algorithm that constructs a DFA recognizing the set of all matching patterns in $O(nm^2)$ time using $O(m)$ space. Notice that we need to examine all possible strings that have an NOI-alignment with any $T_i$ to find $P$. Since the number of inversions in all $T_i$'s is $O(nm^2)$, it would be challenging to design an algorithm that runs faster than $O(nm^2+Cm)$. Since the runtime to find all patterns depends on $C$, it is an interesting open question to establish the lower and the upper bound of $C$. Another future direction is to find frequent patterns from a bio sequence under other biological operations and combined operations.

**Acknowledgments.** This research was supported in part by the Basic Science Research Program through NRF funded by MEST (2012R1A1A2044562) and by the Yonsei University Future-leading Research Initiative of 2014. Kim was supported by NRF Grant funded by the Korean Government (NRF-2013-Global Ph.D. Fellowship Program).

## References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. ACM SIGMOD Record **22**(2), 207–216 (1993)
2. Amir, A., Porat, B.: Pattern matching with non overlapping reversals - approximation and on-line algorithms. In: Cai, L., Cheng, S.-W., Lam, T.-W. (eds.) Algorithms and Computation. LNCS, vol. 8283, pp. 55–65. Springer, Heidelberg (2013)
3. Cantone, D., Cristofaro, S., Faro, S.: Efficient string-matching allowing for non-overlapping inversions. Theoretical Computer Science **483**(29), 85–95 (2013)
4. Chen, Z.Z., Gao, Y., Lin, G., Niewiadomski, R., Wang, Y., Wu, J.: A space-efficient algorithm for sequence alignment with inversions and reversals. Theoretical Computer Science **325**(3), 361–372 (2004)
5. Cho, D.-J., Han, Y.-S., Kang, S.-D., Kim, H., Ko, S.-K., Salomaa, K.: Pseudo-inversion on formal languages. In: Ibarra, O.H., Kari, L., Kopecki, S. (eds.) UCNC 2014. LNCS, vol. 8553, pp. 93–104. Springer, Heidelberg (2014)
6. Cho, D.J., Han, Y.S., Kim, H.: Alignment with non-overlapping inversions and translocations on two strings. Theoretical Computer Science (in press)
7. Harary, F.: Graph Theory. Addison-Wesley series in mathematics. Perseus Books (1994)

8. Ibarra, O.H.: On decidability and closure properties of language classes with respect to bio-operations. In: Murata, S., Kobayashi, S. (eds.) DNA 2014. LNCS, vol. 8727, pp. 148–160. Springer, Heidelberg (2014)
9. Kruskal, Jr., J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. Proceedings of the American Mathematical Society **7**(1), 48–50 (1956)
10. Kum, H.C., Pei, J., Wang, W., Duncan, D.: Approxmap: Approximate mining of consensus sequential patterns. In: Proceedings of the 2nd SIAM International Conference on Data Mining, pp. 311–315 (2003)
11. Liao, V.C.C., Chen, M.S.: DFSP: a Depth-First SPelling algorithm for sequential pattern mining of biological sequences. Knowledge and Information Systems **38**(3), 623–639 (2013)
12. Liao, V.C.C., Chen, M.S.: Efficient mining gapped sequential patterns for motifs in biological sequences. BMC Systems Biology **7**(4), 1–13 (2013)
13. Lupski, J.R.: Genomic disorders: structural features of the genome can lead to DNA rearrangements and human disease traits. Trends in Genetics **14**(10), 417–422 (1998)
14. Sagot, M.-F.: Spelling approximate repeated or common motifs using a suffix tree. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 374–390. Springer, Heidelberg (1998)
15. Schöniger, M., Waterman, M.S.: A local algorithm for DNA sequence alignment with inversions. Bulletin of Mathematical Biology **54**(4), 521–536 (1992)
16. Shallit, J.: A Second Course in Formal Languages and Automata Theory. Cambridge University Press (2008)
17. Vellozo, A.F., Alves, C.E.R., do Lago, A.P.: Alignment with non-overlapping inversions in $O(n^3)$-time. In: Bücher, P., Moret, B.M.E. (eds.) WABI 2006. LNCS (LNBI), vol. 4175, pp. 186–196. Springer, Heidelberg (2006)
18. Wang, K., Xu, Y., Yu, J.X.: Scalable sequential pattern mining for biological sequences. In: Proceedings of the 13th ACM International Conference on Information and Knowledge Management, pp. 178–187 (2004)
19. Wood, D.: Theory of Computation. Harper & Row (1986)
20. Zhu, F., Yan, X., Han, J., Yu, P.S.: Efficient discovery of frequent approximate sequential patterns. In: Proceedings of the 7th IEEE International Conference on Data Mining, pp. 751–756 (2007)