Exact Multiple String Matching Problem for DNA Alphabet

Yi-Kung Shieh, Shyong Jian Shyu and Richard Chia-Tung Lee
Department of Computer Science
National Tsing Hua University, Hsinchu, Taiwan
d9762814@oz.nthu.edu.tw, sjshyu@mail.mcu.edu.tw and rctlee@rctlee.cyberhood.net.tw

Abstract
Given a text $T = t_1 t_2 \ldots t_n$ and a set of patterns $P = \{P_1, P_2, \ldots, P_r\}$, the exact multiple string matching problem (EMSMP) finds the ending positions of all sub-strings in $T$ which is equal to $P_i$, for $1 \leq i \leq r$. We regard all sub-strings in $T$ and patterns in $P$ as data points in an edit distance-based metric space. The data points in $T$ are constructed into a vantage point tree (vp-tree) $\mathcal{C}$. Then, EMSMP can be resolved by searching all points of $P$ in $\mathcal{C}$. We further enhance $\mathcal{C}$ into vpac-tree $\mathcal{E}$ (vp-tree with alliance cut capability), based on which more unnecessary branches might be cut off so that the searching efficiency could be improved. Experiments consisting of long texts and short patterns of DNA alphabet are conducted using the two proposed schemes and $m$-BNDM (the multiple pattern version of the well known BNDM approach). The computational results demonstrate the effectiveness and efficiency of our schemes.

1. Introduction
Consider an alphabet set $\Sigma$ consisting of $\sigma$ symbols and strings whose constituent members are from the symbols of $\Sigma$. Let $T = t_1 t_2 \ldots t_n$ be a text string and $P = \{P_1, P_2, \ldots, P_r\}$ be a set of patterns where $P_i = p_1 p_2 \ldots p_{m_i}$, $m_i = |P_i|$ is the length of $P_i$, and $t_j, p_i \in \Sigma$ for $1 \leq j \leq n, 1 \leq i \leq r$ and $1 \leq j \leq m_i$. Let $T[i, j]$ be the sub-string staring at $i$ and ending at $j$ of $T$ (i.e., $T[i, j] = t_i t_{i+1} \ldots t_j$) where $1 \leq i \leq j \leq n$. The exact multiple string matching problem (EMSPM) is to find all ending positions $\bar{\lambda}_i$'s in $T$ such that $T[\bar{\lambda}_i-m_i+1, \bar{\lambda}_i] = P_i$ (i.e., $t_{\bar{\lambda}_i-m_i+1} t_{\bar{\lambda}_i-m_i+2} \ldots t_{\bar{\lambda}_i} = p_1 p_2 \ldots p_{m_i}$) for $1 \leq i \leq r$. Assume that $T = \{\text{those}, \text{easy}, \text{test}, \text{se}\}$ and $P = \{\{5\}, \{9, 26\}, \{5, 13\}\}$ because $P_1 = \{\text{those}\} = t_1 t_2 t_3 t_4 t_5$, $P_2 = \{\text{easy}\} = t_6 t_7 t_8 t_9 = t_{12} t_{13} t_{14} t_{15}$ and $P_4 = \{\text{se}\} = t_{16} t_{17}$. EMSMP is significant in many applications such as search engine, speech reorganization, computer virus detection, pattern finding in biological sequences, just to name a few.

Aho and Corasick proposed a linear time method to solve this problem [1], referred to as the AC algorithm, which applies the concept of the KMP algorithm [5] to pre-process the patterns in $P$. Commentz-Walter presented another linear time (called the CW) algorithm [3], which combines the Boyer-Moore algorithm [2] with the AC algorithm. The performance of the CW algorithm is better than that of the AC algorithm. Wu and Manber also applied and refined the idea of the Boyer-Moore algorithm for this problem [10], and the resultant WM algorithm is more effective than the CW algorithm. All of these algorithms find the shortest length, denoted as $l_{\text{min}}$, among the patterns in $P$ and set a window $W$ with length $l_{\text{min}}$ to compare with those patterns. They developed several skills to slide the window efficiently along the text to skip unnecessary comparisons. Whenever a match happens in the window, further checking for the rest of $P_i$ (i.e., $P_i[l_{\text{min}}+1, m_i]$) should be compared with the proper sub-string in $T$ to ensure that an exact match does occur.

Later, Navarro and Raffinot proposed the so called backward non-deterministic dawg matching (BNDM) algorithm [7] to solve the exact string matching problem with single pattern (ESMP). It also utilizes the sliding window technologies and has the reputation of being a fast algorithm for ESMP. Navarro and Raffinot further extended the BNDM algorithm to solve EMSMP [8], referred to as $m$-BNDM (where “$m$” is for multiple string).

In this study, we propose two schemes relying on the vantage point tree (vp-tree) [11] to solve EMSMP for the DNA alphabet. Surely, the proposed schemes are applicable for any alphabet. Originally, the vp-tree was devised to organize the data points in a metric space in order to answer the range query problem. Our first scheme regards the sub-strings in $T$ and the patterns in $P$ as the data points in an edit distance-based metric space. The data points in $T$ would be constructed into vp-tree $\mathcal{C}$ and those in $P$ are regarded as queries. EMSMP can thus be solved by searching all points of $P$ in $\mathcal{C}$. The second scheme further explores the possibility to cut more unnecessary branches in $\mathcal{C}$ to enhance the searching efficiency.

The rest of this paper is organized as follows. In Section 2, we briefly introduce the concept of the $m$-BNDM algorithms, and the structure of vp-tree in dealing with the range query problem. In Section 3, we propose two schemes relying on vp-tree for solving EMSMP. Section 4 compares the experimental results of the proposed schemes against those of $m$-BNDM on several simulated DNA datasets. Finally, Section 5 gives some concluding remarks.
2. Previous Study

The elegant m-BNDM, which is the comparison basis of our algorithms, is briefly introduced in Section 2.1. The originality of vp-tree, based on which the text and the patterns in the problem considered would be transformed, is examined in Section 2.2.

2.1 m-BNDM

m-BNDM first finds the shortest length \( l_{\text{min}} \) among all patterns. Let the prefix with length \( l_{\text{min}} \) of \( P_i \) be \( P'_i \). It reverses all \( P'_i \)'s which are denoted as \( P''_i \)'s, and then concatenates them to a long pattern \( P' \) (i.e., \( P' = P''_1 P''_2 \ldots P''_{n} \)). An auxiliary \( 0/1 \sigma \times \|P'\|/\|\omega \) storage \( B \) is used to record the positions of all characters of \( \Sigma \) in \( P' \). Specifically, if the character of \( \ell \) position in \( P' \) is \( x \), the \( \ell \)th bit of \( B[x] \) is 1; otherwise 0. Note that we use \( \|P'\|/\|\omega \) computer words to record \( B[x] \), where \( \omega \) is the size of a word in computer (i.e. 32 or 64 bits). It searches text \( T \) by considering window \( W \) at position \( i \) where \( W = T[i, i+l_{\text{min}}-1] \) and \( W = T[1, l_{\text{min}}] \) initially. The purpose is to find the longest suffix in \( W \) which is equal to the prefix of at least one pattern. There are two variables \( D \) and \( last \) where the size of \( D \) is also \( \|P'\|/\|\omega \) computer words. If \( D \neq 0 \), we have to move \( W \) with \( last \) steps to the right. Before each movement of \( W \), \( D = 1 \|P'\| \) and \( last = l_{\text{min}} \). Then the characters in \( W \) are read from right to left and one by one. When the character \( y \) of \( j \)th position in \( W \) is read, \( D' = D & B[y] \) where \& is the AND operation, which means the information of the character \( y \) in \( j \)th position is superposed into \( D' \). Let \( DF = (10^{l_{\text{min}}-1})' \). If \( D' \& DF \neq 0 \), there is a suffix with length \( l_{\text{min}}-j+1 \) which is equal to the prefix of at least one pattern. Hence, if \( D' \& DF \neq 0 \) and \( j = 1 \), the length of suffix is \( l_{\text{min}} \). This applies that the string in \( W \) is equal the suffix of some pattern(s). By checking whether \( D'0(\|P'\|-l_{\text{min}}-1)10^{l_{\text{min}}-1})' \) is zero for \( 1 \leq s \leq r \), we know the prefix belongs to \( P_i \); further checking whether \( P_i[l_{\text{min}}, m] = T[1+l_{\text{min}}, i+l_{\text{min}}-1] \) leads us to report an exact match with ending position \( i+l_{\text{min}}-1 \). If \( D' \& DF \neq 0 \) and \( j \neq 1 \), last is updated to be \( j-1 \). After testing \( D' \& DF \), we set \( D = (D' \times s < k) \& (10^{l_{\text{min}}-1})' \) where \( \times \) is the left shift operation. If \( D \neq 0 \), the character of \( (j-1) \)th is read and it continuously utilizes the above method. Otherwise, in order to align the suffix (with length \( l_{\text{min}}-last \)) with the prefix of a pattern, \( W \) is moved \( last \) steps to the right. In short, m-BNDM reads characters in \( W \), tests \( D' \) to collect matches, operates \( D \) and moves \( W \) until all text has been considered to resolve EMSMP.

2.2 Vp-tree

A metric space is an order pair \((A, d)\) where \( A \) is a set of data points and \( d \) is a metric function on \( A \) such that for any \( x, y, z \in A \), the following holds [9]:

(i) \( d(x, y) = d(y, x) \);
(ii) \( d(x, y) = 0 \) when \( x = y \);
(iii) \( 0 < d(x, y) < \infty \) when \( x \neq y \);
(iv) \( d(x, y) \leq d(x, z) + d(z, y) \) (triangle inequality).

The vp-tree, initially invented by Yiannis [11], aims at organizing all input data points in a metric space into a balanced binary tree to facilitate the range query problem. Given a query point \( q \), a distance \( \gamma \) and a set of data points \( X = \{x_1, x_2, \ldots, x_n\} \) from a metric space with function \( d \), the range query problem finds out all data points \( x_i \)'s such that the distance between \( q \) and \( x_i \) is no more than \( \gamma \) (i.e., \( d(q, x_i) \leq \gamma \)) for \( x_i \in X \). Once vp-tree \( \mathcal{C} \) with respect to \( X \) is built, the query specified by \((q, \gamma)\) could be easily answered by searching points in \( \mathcal{C} \) whose distance from \( q \) is within the range \( \gamma \).

The construction of vp-tree \( \mathcal{C} \) with respect to \( X \) is a recursive process involving a ball partition procedure. This procedure starts from choosing a point, say \( x_i \), which is called the vantage point, randomly from \( X \). All distances between \( x_i \) and those in \( X \setminus \{x_i \} \) (i.e., \( d(x_i, x_j) \)’s for \( x_j \in X \setminus \{x_i \} \)) are computed. Let \( e \) be the median of all \( d(x_i, x_j) \)’s. Then, \( X \setminus \{x_i \} \) would be evenly partitioned into \( X_e \) and \( X_{\neg e} \) such that

\[
X_e = \{x_j \mid d(x_i, x_j) \leq e \text{ for } x_j \in X \setminus \{x_i \} \}
\]

and

\[
X_{\neg e} = \{x_j \mid d(x_i, x_j) > e \text{ for } x_j \in X \setminus \{x_i \} \}.
\]

Conceptually, such data decomposition can be regarded as the partitioning result of ball \( B(x_i, e) \) centered at vantage point \( x_i \), with radius \( e \) such that \( X_e \) and \( X_{\neg e} \) locate inside and outside \( B \), respectively. That’s why we call it the ball partition. After \( X_e \) and \( X_{\neg e} \) are recursively constructed as left and right sub-trees \( \mathcal{C}_e \) and \( \mathcal{C}_{\neg e} \), respectively. We could build a node \( N \) consisting of \( \{x_i, e, \mathcal{C}_e, \mathcal{C}_{\neg e}\} \) to be the root of the vp-tree \( \mathcal{C} \) with respect to \( X \). Surely, the recursion returns an empty tree when the input set of data points becomes empty. Since at each node the data points are evenly partitioned, \( \mathcal{C} \) is a balanced binary tree.

When facing a query \((q, \gamma)\), we simply search \( \mathcal{C} \) as follows. The search begins from the root of \( \mathcal{C} \). Let the node being searched be \( N = (x_i, e, \mathcal{C}_e, \mathcal{C}_{\neg e}) \). We compute the distance between vantage point \( x_i \) and query point \( q \): \( q = d(x_i, q) \). If \( q \leq \gamma \), \( x_i \) is surely one of the solution for query \((q, \gamma)\). Then, \( \eta \) is compared against \( e+\gamma \) and \( e-\gamma \). If \( \eta > e+\gamma \), \( \mathcal{C}_e \) can be cut (since there is no solution candidate) and only \( \mathcal{C}_{\neg e} \) would be further searched; if \( \eta < e-\gamma \), \( \mathcal{C}_{\neg e} \) is pruned away and only \( \mathcal{C}_e \) is searched; otherwise \( e-\gamma \leq \eta \leq e+\gamma \), both \( \mathcal{C}_e \) and \( \mathcal{C}_{\neg e} \) would be searched. The search goes onto \( \mathcal{C}_e \) or\( \mathcal{C}_{\neg e} \) in a recursive way and all the vantage points in the searched nodes whose
3. Proposed Schemes for Solving EMSMP

Our strategy for coping with EMSMP is constructing the substrings in text \( T \) as vp-tree \( \mathcal{C} \) and then reporting the matches of patterns in \( P \) by searching them in \( \mathcal{C} \). In essence, we measure the distance between two strings by way of the edit distance [6]. Let \( a \) and \( b \) be two strings. The edit distance between \( a \) and \( b \), denoted as \( \delta(a, b) \), is the minimum number of edit operations needed to convert \( b \) into \( a \) where an edit operation can be an insertion, deletion or substitution of a character. Thus, pattern \( p \) is found in text \( T \) if and only if \( \delta(p, t) = 0 \) for some sub-string \( t \) in \( T \) with \( ||p|| = ||t|| \). The edit distance function is metric because it satisfies the aforementioned conditions (i)-(iv) in Section 2.2 [9].

Therefore, applying the idea of vp-tree for indexing data points with edit distance in a metric space is promising in our research. In the following, we present a simple realization of mapping text \( T \) into vp-tree \( \mathcal{C} \) in Section 3.1. The algorithm for searching \( P \) in \( \mathcal{C} \) for \( 1 \leq i \leq r \) is designed in Section 3.2. A brilliant refinement on the vp-tree with additional branch-cutting capability to resolve EMSMP are proposed in the following section.

3.1 Basic vp-tree

Consider EMSMP with text \( T \) and pattern set \( P \). Let \( l_{\text{min}} \) be the shortest length among the set of patterns, i.e., \( l_{\text{min}} = \min(||P_1||, ||P_2||, \ldots, ||P_r||) = (m_1, m_2, \ldots, m_r) \). Let \( x_j \) denote the sub-string in \( T \) starting at \( j \) and ending at \( j+l_{\text{min}}-1 \), i.e., \( x_j = T[j, j+l_{\text{min}}-1] \) for \( 1 \leq j \leq n-l_{\text{min}}+1 \). Let \( X = \{x_1, x_2, \ldots, x_{n-l_{\text{min}}+1} \} \) be the data point set (consisting of substrings in \( T \) of length \( l_{\text{min}} \)) and \( Q = \{q_1, q_2, \ldots, q_l \} \) where \( q_i = P_i[l_{\text{min}}] \) for \( 1 \leq i \leq r \) be the query point set (containing prefixes in \( P \) of length \( l_{\text{min}} \)). We shall build vp-tree \( \mathcal{C} \) for \( X \) and search all members of \( Q \) in \( \mathcal{C} \).

For each pair of \( x_j \) and \( q_i \), if \( \delta(q_i, x_j) \neq 0 \) (or \( x_j \neq q_i \)), there is no chance for pattern \( P_i \) finds a match with the substring staring at \( j \) in \( T \). On the other hand, \( P_i \) might attain a match in \( T \) staring at \( j \) (i.e., \( P_i = T[j, j+m_i-1] \)) only if \( \delta(q_i, x_j) = 0 \). These are so informative that we address them as a property as follows.

**Property 1.** For each pair of \( x_j \) and \( q_i, 1 \leq j \leq n-l_{\text{min}}+1 \) and \( 1 \leq i \leq r \),
1. if \( \delta(q_i, x_j) \neq 0 \), \( P_i \neq T[j, j+m_i-1] \);
2. if \( \delta(q_i, x_j) = 0 \), \( P_i = T[j, j+m_i-1] \), only if \( \delta(q_i, x_j) = 0 \).

Our basic idea is clear now. The query of \( q_i \) in vp-tree \( \mathcal{C} \) results in all possible match candidates in \( T \) and, meanwhile, filters out impossible sub-strings aggressively for \( 1 \leq i \leq r \). Surely, before we report that \( P_i \) receives a match in \( T \) starting at position \( j \) (ending at \( j+m_i-1 \)) when \( \delta(q_i, x_j) = 0 \), we need to verify whether \( P_i[l_{\text{min}}+1, m_i] = T[j+l_{\text{min}}, j+l_{\text{min}}+m_i-1] \), which is called the tail checking.

To construct vp-tree \( \mathcal{C} \) for \( X = \{x_1, x_2, \ldots, x_{n-l_{\text{min}}+1} \} \), we perform a refined ball partition procedure first. A data point, namely \( x_j \), is randomly selected from \( X \) to be the vantage point. We delete \( x_j \) from \( X \) and initialize a set \( V \) to hold \( x_j \) (i.e., \( X = X \setminus \{x_j\} \) and \( V = \{x_j\} \)). All edit distances between \( x_j \) and \( x_k \)’s \( \delta(x_j, x_k) \) are computed for \( x_k \in X \) and once we find \( \delta(x_j, x_k) = 0 \), which means \( x_k \) is the same as the vantage point \( x_j \) (or equivalently, a match candidate occurs), we remove \( x_j \) from \( X \) and add it into \( V \). That is, \( V \) collects all data points that are the same as the vantage point. In EMSMP, it is quite often for a substring to appear several times in \( T \). It is noticed that no such \( V \) exists in the original design of vp-tree. Let \( e \) be the median of all \( \delta(x_j, x_k) \)’s. We partition \( X \) evenly into two subsets \( X_L \) and \( X_R \) such that

\[
X_L = \{a \mid \delta(x_j, a) \leq e \text{ and } a \in X\}
\]

\[
X_R = \{b \mid \delta(x_j, b) > e \text{ and } b \in X\}
\]

where \( |X_L| - |X_R| \leq 1 \).

After the refined ball partition, we recursively build sub-trees \( \mathcal{C}_L \) and \( \mathcal{C}_R \) with respect to \( X_L \) and \( X_R \). The recursion returns an empty tree when the size of the input data points is 0. As long as \( \mathcal{C}_L \) and \( \mathcal{C}_R \) are built, we create a node \( N \) consisting of \( V \), \( e \), \( \mathcal{C}_L \), and \( \mathcal{C}_R \) as the root node of \( \mathcal{C} \). The basic structure of node \( N \) in \( \mathcal{C} \) is illustrated in Figure 1. Note that, without ambiguity, we refer to \( \mathcal{C}(\mathcal{C}_L \text{ or } \mathcal{C}_R) \) as a tree (sub-tree) or its corresponding pointer in physical implementation interchangeably, and \( N (N_L \text{ or } N_R) \) as its root node consisting of \( V \) and \( e \) (\( V_L \) and \( e_L \) or \( V_R \) and \( e_R \), if not empty).
The whole idea of constructing the vp-tree with respect to $X$ ($T$) for EMSMP is formally presented in Algorithm 1. The computation of the edit distance between two strings with length $l_{\text{min}}$ is in time $O(l_{\text{min}}/\omega \times l_{\text{min}})$ by applying the bit-vector algorithm [4], were $\omega$ is the size of a computer word. The determination of median $e$ can be achieved in $O(n)$ using the median of median algorithm (in the $k$th selection problem) or the bucket-sort. The height of vp-tree is $O(\log n)$ owing to the evenly partitioning at each node recursively, which causes it a balanced binary search tree. Hence, the time complexity of Algorithm 1 is $O(l_{\text{min}}/\omega \times l_{\text{min}} \times n \log n)$.

**Algorithm 1. Constructing a vp-tree**

**Input:** Data point set $X = \{x_1, x_2, \ldots, x_{n-\text{class}+1}\}$ corresponding to text $T = t_1t_2 \ldots t_n$

**Output:** Vp-tree $\mathcal{T}$ with respect to $X$ ($T$)

**RefinedBall Partition ($X, x_i$)**
1. $X = X|x_i|$ and $V = \{x_i\}$
2. for (each $x_j \in X$) do
   - Calculate $\delta(x_i, x_j)$
   - if ($\delta(x_i, x_j) = 0$) then
     - $X = X|x_j|$ and $V = V \cup \{x_j\}$
   - endfor
3. $e$ = median of $\{\delta(x_i, x_j) \mid x_j \in X\}$
4. Evenly divide $X$ into $X_L$ and $X_R$ such that $X_L = \{x_i \mid \delta(x_i, x_j) \leq e \land x_j \in X\}$ and $X_R = \{x_i \mid \delta(x_i, x_j) > e \land x_j \in X\}$ where $|X_L| - |X_R| \leq 1$
5. return $(V, e, X_L, X_R)$

**ConstructVP-tree ($X$)**
1. if ($X = \emptyset$) then return an empty vp-tree
2. Arbitrarily choose a point from $X$, say $x_i$, to be the vantage point
3. $(V, e, X_L, X_R) = \text{RefinedBallPartition} (X, x_i)$
4. $\mathcal{T}_L = \text{ConstructVP-tree} (X_L)$
5. $\mathcal{T}_R = \text{ConstructVP-tree} (X_R)$
6. Create node $N = (V, e, \mathcal{T}_L, \mathcal{T}_R)$ with pointer $\mathcal{T}$
7. return $\mathcal{T}$

---

**Example 1**

Assume that there are 15 points, $X = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\}$ as shown in Figure 2(a) in which every point represents a sub-string in $T$. The final result of refined ball partitions is illustrated in Figure 2(b) and vp-tree $\mathcal{T}$ with respect to $X$ is presented in (c). The construction details are exhibited and explained in Appendix A. Note that in root nor $N_1$ the vantage point is $a$ and median distance $e_1 = \delta(a, e) = \delta(a, j) = \delta(a, m)$; that is, there are three points e, j and m whose distance from a are the same. To highlight these three points, we use red triangles to represent them in Figure 2(b).

### 3.2 Searching in basic vp-tree

To answer whether a pattern, say $P_i$ where $1 \leq i \leq r$, receives matches in $T$, we regard $P_i[1, l_{\text{min}}]$, the prefix with length $l_{\text{min}}$ of $P_i$, as a query point $q_i$ and search $q_i$ in $\mathcal{T}$ constructed by Algorithm 1. For all the possible match candidates, we ought to perform a trail checking procedure one by one. Note that our goal is to find exact matches, thus there is no range distance $\gamma$ (adopted in a traditional range query problem) for a query.

Since $\mathcal{T}$ is a balanced binary search tree, searching $q_i$ in $\mathcal{T}$ is quite easy. Starting from the root node $N$ (consisting of $(V, e, \mathcal{T}_L, \mathcal{T}_R)$), we compute $\eta = \delta(x_i, q_i)$ for a certain $x_i \in V$. On condition that $\eta = 0$ (that is, $x_i = T[v, v+l_{\text{min}}-1] = P_i[1, l_{\text{min}}] = q_i$), we realize that all the points in $V$ are solution candidates by Property 1 (b). A further tail checking procedure for each candidate $x_i \in V$ goes on: If $P_i[l_{\text{min}}+1, m] = T[v+l_{\text{min}}-1]$ position $v+l_{\text{min}}-1$ of $T$ is in the solution set; otherwise, ignore $x_i$. When all $x_i$’s in $V$ are checked, the search corresponding to this node can be terminates (that is, no need to search the sub-trees since all points equal to $q_i$ in the tree rooted at $N$ have already been collected in $V$ during the construction of $\mathcal{T}$).
On the contrary, when \( \eta \neq 0 \), we compare \( \eta \) with \( e \). If \( \eta > e \), left sub-tree \( \mathcal{C}_l \) could be cut off; while \( \eta < e \), right sub-tree \( \mathcal{C}_r \) could be pruned away. We call this a \( b \)-cut owing to the merit of binary search. Property 2 emphasizes this.

**Property 2.** If \( \eta (= \Delta(x_j, q_i)) > e \) in \( N = (V, e, \mathcal{C}_l, \mathcal{C}_r) \), \( \mathcal{C}_r \) can be a \( b \)-cut; and if \( \eta < e \), \( \mathcal{C}_l \) can be a \( b \)-cut.

In short, if \( \Delta(x_j, q_i) \leq e \), the left sub-tree \( \mathcal{C}_l \) will be searched; in addition, if \( \Delta(x_j, q_i) \geq e \), the right sub-tree \( \mathcal{C}_r \) will be searched in a recursive manner. Surely, the search returns an empty set when the sub-tree searched becomes empty.

The details of matching patterns \( P_1, P_2, \ldots, P_n \) in \( P \) in text \( T \) via searching \( q_i \) in \( \mathcal{C}_r \) is shown in Algorithm 2 where \( 1 \leq i \leq r \). Because the height of \( \eta \)-tree is \( O(\log n) \), the expected time complexity of Algorithm 2 is \( O(\log \eta n) \).

**Algorithm 2.** Searching patterns in a \( \eta \)-tree

**Input:** Text \( T = t_1t_2 \ldots t_r \), \( \mathcal{C}_l \) with respect to \( X(T), l_\text{min} \) and pattern set \( P = \{P_1, P_2, \ldots, P_n\} \)

**Output:** \( S = \{S_1, S_2, \ldots, S_k\} \) in which \( S_i \) consists of the ending positions of the substrings in \( T \) which are exactly equal to \( P_i \) for \( 1 \leq i \leq r \)

1. if \( (\mathcal{C}_l = \emptyset) \) then return \( \emptyset \)
2. Obtain \( (V, e, \mathcal{C}_l, \mathcal{C}_r) \) from the node pointed by \( \mathcal{C}_l \)
3. Let \( q_i = P_i[1, l_\text{min}] \)
4. Choose any point, say \( x_j \), in \( V \) as the vantage point
5. Compute \( \eta = \Delta(x_j, q_i) \) and set \( S = \emptyset \)
6. a. if \( (\eta > 0) \) then
6.a.1. for (each \( x_j \in V \)) do // tail searching
6.a.2. if \( T[j+l_{\text{min}}+1, \ldots, l_{\text{min}}+1] = P_i[l_{\text{min}}+1, l_{\text{min}}+1] \) then
   \[ S = S \cup \{j+l_{\text{min}}+1\} \]
7. endfor
6.b. else
6.b.1. if \( (\Delta(x_j, q_i) \leq e) \) then
    \[ S = S \cup \text{QueryVP}(T, \mathcal{C}_l, l_{\text{min}}, P_i) \]
6.b.2. if \( (\Delta(x_j, q_i) \geq e) \) then
    \[ S = S \cup \text{QueryVP}(T, \mathcal{C}_r, l_{\text{min}}, P_i) \]
8. end

**SolveEMSMP(T, P)**

1. \[ X = \{x_1, x_2, \ldots, x_{l_{\text{max}}-1}\} \] where \( x_j = T[j, j+l_{\text{min}}-1] \) for \( 1 \leq j \leq l_{\text{max}}-1 \)
2. \( \mathcal{C}_l = \text{ConstructVP-tree}(X) \)
3. \( l_{\text{min}} = \min \{ ||P_i|| | P_i \in P \} \)
4. for (each \( i, 1 \leq i \leq r \)) do
   \[ S_i = \text{QueryVP}(T, \mathcal{C}_l, l_{\text{min}}, P_i) \]

endfor

5. return \( S = \{S_1, S_2, \ldots, S_r\} \)

The effectiveness of collecting all data points having the same distance with the selected vantage point in \( V \) for each node is evident in Algorithm 2. Whenever \( \Delta(x_j, q_i) = 0 \) occurs for some \( x_j \in V \) in node \( N \), we stop further searching \( N \)'s descendants, since all possible solution candidates in the tree rooted at \( N \) are assessable in the current \( V \).

**Example 2** Following Example 1 where vp-tree \( \mathcal{C}_r \) is shown in Figure 2, consider a query \( q_i \) that is exactly equal to point \( j \) (i.e., \( q_i = j \)). We search \( \mathcal{C}_l \) and start from root \( N_1 \). In \( N_1 = ([a], e, \mathcal{C}_l', \mathcal{C}_r') \), \( \Delta(a, q_i) \neq 0 \) and \( \Delta(a, q_i) = e_1 \) where \( e_1 \) is the median distance of \( N_1 \). Hence, we have to search both \( \mathcal{C}_l' \) and \( \mathcal{C}_r' \) (rooted at \( N_2 \) and \( N_9 \), respectively). In \( N_2 \), because \( \Delta(b, q_i) \neq 0 \) and \( \Delta(b, q_i) = e_2 \), both \( \mathcal{C}_l' \) and \( \mathcal{C}_r' \) need to be considered. In \( N_9 \), since \( \Delta(d, q_i) \neq 0 \) and \( \Delta(d, q_i) = e_3 \), only \( N_7 \) (but not \( N_8 \) by Property 2) would be searched. In leaf \( N_7 \), we find that \( \Delta(j, q_i) = 0 \) and no more child node would be searched. Hence, there is no solution in sub-tree \( \mathcal{C}_r' \). Let us consider \( \mathcal{C}_l' \). In \( N_8 \), owing to \( \Delta(e, q_i) \neq 0 \) and \( \Delta(e, q_i) = e_4 \), \( N_1 \) and \( N_2 \) need to be considered. In leaf \( N_7 \), we find that \( \Delta(j, q_i) = 0 \) so that \( S = \{j\} \) and stop searching any descendant node of \( N_1 \). In leaf \( N_8 \), \( \Delta(j, q_i) = 0 \) and no child node would be searched. By now, the whole left sub-tree \( \mathcal{C}_r' \) has been searched and we go on searching \( \mathcal{C}_r' \) rooted at \( N_9 \). In \( N_9 \), \( \Delta(c, q_i) \neq 0 \) and \( \Delta(c, q_i) < e_5 \) so that only sub-tree \( \mathcal{C}_r'' \) (but not \( \mathcal{C}_r' \) by Property 2) need to be considered. In \( N_10 \), since \( \Delta(f, q_i) \neq 0 \) and \( \Delta(f, q_i) > e_10 \), only \( \mathcal{C}_r'' \) (instead of \( \mathcal{C}_r' \)) need to be considered. In leaf \( N_{12} \), \( \Delta(m, q_i) \neq 0 \) and no more child node could be searched. Finally, we obtain \( S_1 = \{j\} \). Note that in this case ten out of the fifteen nodes are searched.

3.3 Vp-tree with alliance cut

When querying \( q_i \) in \( \mathcal{C}_l \) rooted at \( N \), we may encounter a situation that both \( \mathcal{C}_l \) and \( \mathcal{C}_r \) need to be queried further. For instance, both sub-trees in \( N_1, N_2 \) and \( N_6 \) of Example 2 ought to be searched owing to \( \Delta(a, q_i) = e_1 \), \( \Delta(b, q_i) = e_2 \) and \( \Delta(e, q_i) = e_6 \), respectively. Concerning such situation \( (\Delta(x_j, q_i) = e) \) in \( N \) consisting \( V \) and \( e \) where \( x_j \in V \), we shall introduce a skill, referred to as the \textit{alliance cut}, to determine whether \( N_j \)'s (or \( N_k \)'s) sub-trees could be cut because there is no chance to gather exact matches any more.

Let us observe a feature on the distance relationship in our scenario. Consider two data points \( a \) and \( b \), and one vantage point \( v \). On condition that \( \Delta(v, a) \neq \Delta(v, b) \), it is impossible to have \( a = b \). On the contrary, \( a = b \) only if \( \Delta(v, a) = \Delta(v, b) \). We summarize these in Property 3.
Property 3. For any pair of data points $a$ and $b$, and vantage point $v$:
(a) if $\delta(v, a) \neq \delta(v, b)$, $a \neq b$; and
(b) $a = b$ only if $\delta(v, a) = \delta(v, b)$.

Figure 3 illustrates some examples for Property 3:
(a) $\delta(v, a) \neq \delta(v, b)$, thus $a \neq b$; (b) $\delta(v, a) = \delta(v, b)$, but $a \neq b$, and (c) $a = b$ so that $\delta(v, a) = \delta(v, b)$.

![Figure 3. Some examples for Property 3](image)

The concept of the proposed alliance cut (a-cut, for short) utilizes Property 3 and is informally explained via Examples 1 and 2 (see Figure 2). When searching $q_i (= j)$ in $N_i$ with vantage point $a$ and medium distance $e_i$, we find thee points $e$, $j$ and $m$ such that $\delta(a, e) = \delta(a, j) = \delta(a, m) = e_i$. By Algorithm 2, we know that both sub-trees $\mathcal{C}_e^i$ and $\mathcal{C}_m^i$ should be searched. However, we could obtain more clues from this case: (1) These three points are solution candidates since they may be equal to $q_i$ by Property 3 (b); and (2) the other points would not be the same as $q_i$ any more by Property 3 (a). Let us check where these three points locate after the revised ball partition in $N_i$. Observing $\mathcal{C}_e^i$ rooted at $N_9$, we know that $N_9$ classifies $e$ and $j$ into $\mathcal{C}_e^i$. It means that there is no need to search $\mathcal{C}_m^i$ (in which no solution candidate exists). In addition, observing $\mathcal{C}_m^i$ rooted at $N_9$, we find that $m$ is classified into $\mathcal{C}_m^i$. It means that there is no need to search $\mathcal{C}_e^i$ (since there is no solution candidate). Algorithm 2 could detect the latter (no need to search $\mathcal{C}_m^i$) by b-cut at $N_9$, but it could not explore the former (so that $\mathcal{C}_e^i$ would be searched in Algorithm 2). Our a-cut aims at cutting off such $\mathcal{C}_m^i$ and $\mathcal{C}_e^i$.

Let $N$ denote a certain node in $\mathcal{C}$, $N'$ be its father, $N''$ be its brother, $\mathcal{C}_e$ and $\mathcal{C}_b$ be its two sub-trees rooted at $N_2$ and $N_3$ where $N = (V, e, \mathcal{C}_e, \mathcal{C}_b), N' = (V', e', \mathcal{C}_e', \mathcal{C}_b')$ and $N'' = (V'', e'', \mathcal{C}_e'', \mathcal{C}_b'')$. The alliance cut decision in $N$ ($N''$) about cutting $\mathcal{C}_e$ or $\mathcal{C}_b$ ($\mathcal{C}_e'$ or $\mathcal{C}_b'$) relies on $N$’s detection about $E \neq \emptyset$ where $E = \{ x' | \delta(x', q) = e' \text{ for } x' \in \mathcal{X}' \}$. We call it an alliance cut for the reason that father’s ($N'$’s) knowledge is alliance with its son $N$ ($N''$) and after $N$ ($N''$) finishes the revised ball partition, such knowledge is capable of judging whether some sub-tree(s) $\mathcal{C}_e$ or $\mathcal{C}_b$ ($\mathcal{C}_e'$ or $\mathcal{C}_b'$) may be cut off.

Property 4. If $\delta(x', q) = e'$ and $E' \neq \emptyset$ in $N' = (V', e', \mathcal{C}_e, \mathcal{C}_b)$ with respect to $X'$, $\mathcal{C}_e$ or $\mathcal{C}_b$ ($\mathcal{C}_e'$ or $\mathcal{C}_b'$) in $N$ ($N''$) which contains no member of $E'$ can be a-cut by the alliance of $(x', x_e)$ where $E' = \{ x' | \delta(x', q) = e' \text{ for } x' \in \mathcal{X}' \}$.

Figure 4 gives a general case that a-cuts do occur where (a) depicts a situation that six points $x_4$’s (in red) are detected with $\delta(x_i, x_4) = e'$ (i.e. $|E'| = 6$) in $N'$ and (b) tells that three of the six are decomposed into $\mathcal{C}_e$ in $N$ (in red triangles) and the other three into $\mathcal{C}_b$ in $N'$. Then, $\mathcal{C}_e$ and $\mathcal{C}_b'$, which contain no member of $E'$, can be pruned by a-cuts in $N$ from the alliance of vantage points $x_i$ and $x'_i$.

![Figure 4. Example of alliance cuts in vp-tree](image)
returns an empty node on condition that the input data point set is empty. After the construction of \( \mathcal{C}_L \) and \( \mathcal{C}_R \), we establish \( N = (V, e, \mathcal{C}_L, \mathcal{C}_R, (G_L, G_R)) \) as the root of \( \mathcal{C} \). Initially, owing to no parent for the root node, \( E \) at the root is merely an empty set.

Our approach for constructing vpac-tree is shown in Algorithm 3.

### Algorithm 3. Constructing a vpac-tree

**Input:** Data point set \( X = \{x_1, x_2, \ldots, x_{\text{size}}\} \) corresponding to text \( T = t_1t_2 \ldots t_n \) and \( \text{equal-distance-to-e set } E \) (initially \( E = \emptyset \)).

**Output:** Vpac-tree \( \mathcal{C} \) of \( X \) in \( T \).

**ConstructVPAC-tree** \((X, E)\): 

1. if \( (X = \emptyset) \) then return an empty vp-tree

2. Arbitrarily choose a point from \( X \), say \( x_r \), to be the vantage point

3. \((V, e, X_L, X_R) = \text{RefinedBallPartition}(X, x_r)\)

4. \( G_L \neq 0 \) and \( G_R = 0 \)

5. if \( (x_r \in E) \) for any \( x_r \in X_R \) then \( G_L = 1 \)

6. if \( (x_r \in E) \) for any \( x_r \in X_L \) then \( G_R = 1 \)

7. \( E_L = \emptyset \) and \( E_R = \emptyset \)

8. for each \( a, x_r \in X_L \) do

   if \( (\delta(x_r, x_r) = e) \) then \( E_L = E_L \cup \{x_r\} \)

9. for each \( b, x_r \in X_R \) do

   if \( (\delta(x_r, x_r) = e) \) then \( E_R = E_R \cup \{x_r\} \)

10. \( \mathcal{C}_L = \text{ConstructVPAC-tree}(X_L, E_L) \)

11. \( \mathcal{C}_R = \text{ConstructVPAC-tree}(X_R, E_R) \)

12. Create a node \( N = (V, e, \mathcal{C}_L, \mathcal{C}_R, G) \) with pointer \( \mathcal{C} \) where \( G = (G_L, G_R) \)

13. return \( \mathcal{C} \)

### 3.4 Searching in basic vpdc-tree

When dealing with a query point \( q_i (= P_i[1, l_{\text{min}}]) \) at node \( N \) containing \( V \) and \( e \) in \( \mathcal{C} \), we simply compute \( \eta = \delta(x_r, q_i) \) where \( 1 \leq i \leq r \) and \( x_r \in \text{V} \). If \( \eta = 0 \), a subsequent tail checking is performed for each member in \( V \) to gather matches into the solution set. The search in the sub-tree rooted at \( N \) ends and the solution set is returned (just as in the basic vp-tree). While \( \eta \neq 0 \), we test whether \( \eta = e \): If yes (a case that possible a-cuts may occur in its sons), a flag of \( \text{equal-distance-to-e-between-query-vantage} \) is set, i.e., \( edeqv = 1 \); otherwise \( edeqv = 0 \). This flag would be passed to \( N \)’s sub-trees \( \mathcal{C}_L \) and \( \mathcal{C}_R \) rooted at \( N_L \) and \( N_R \) which rename it as “p_edeqv”. Of course, \( N \) receives such \( p_{\text{edeqv}} \) from its father \( N^j \). Note that in the root node, \( p_{\text{edeqv}} = 0 \) since it has no parent.

On condition that \( N \) gets \( p_{\text{edeqv}} = 1 \) from its father \( N^j \), \( G = (G_L, G_R) \) becomes useful. If \( G_L = 0 \), \( \mathcal{C}_L (\mathcal{C}_R) \) can be a-cut directly by Property 4. In fact, \( \mathcal{C}_L (\mathcal{C}_R) \) would only be searched when \( \eta \leq e \) (it is not b-cut by vantage point \( i \) in \( N \) and \( i_L \) in \( N^j \)).

Otherwise \( (p_{\text{edeqv}} = 0) \), the conventional vantage point tests (as in Algorithm 2) are followed to decide which sub-trees ought to be searched. Algorithm 4 shows how to search pattern \( P_i \) in a vpac-tree for \( 1 \leq i \leq r \).

### Algorithm 4. Searching a pattern in a vpac-tree

**Input:** Text \( T \), vpac-tree \( \mathcal{C} \) of \( T \), length \( l_{\text{min}} \), pattern \( P_i \) and flag \( p_{\text{edeqv}} \) (initially, \( p_{\text{edeqv}} = 0 \)).

**Output:** \( S \) consisting of the ending positions of the substrings in \( T \) which are exactly equal to \( P_i \).

**QueryVPAC-tree**\((T, \mathcal{C}, l_{\text{min}}, P_i, p_{\text{edeqv}})\): 

1. if \( (\mathcal{C} = \emptyset) \) then return \( \emptyset \)

2. Obtain \( (V, e, \mathcal{C}_L, \mathcal{C}_R, (G_L, G_R)) \) in the node pointed by \( \mathcal{C} \)

3. Let \( q_i = P_i[1, l_{\text{min}}] \)

4. Choose any point, say \( x_r \), in \( V \) as the vantage point

5. Compute \( \eta = \delta(x_r, q_i) \) and set \( S = \emptyset \)

6.a if \( (\eta = 0) \) then

6.a.1 for each \( x_r \in V \) do // tail checking

   if \( (T|v+m_{\text{min}}, v+m_{\text{min}})^1 = P_i[l_{\text{min}}+1, m_i]) \) then

   \( S = S \cup (v+m_{\text{min}}-1) \)

endfor

6.b else

6.b.1 if \( (\eta = e) \) then \( edeqv = 1 \) else \( edeqv = 0 \)

6.b.2.a if \( (p_{\text{edeqv}} = 1) \) then

6.b.2.a.1 if \( (\eta \leq e \text{ and } G_L = 1) \) then

   \( S = S \cup \text{QueryVPAC-tree}(T, \mathcal{C}_L, l_{\text{min}}, P_i, p_{\text{edeqv}}) \)

6.b.2.a.2 if \( (\eta \geq e \text{ and } G_R = 1) \) then

   \( S = S \cup \text{QueryVPAC-tree}(T, \mathcal{C}_R, l_{\text{min}}, P_i, p_{\text{edeqv}}) \)

endif

6.b.2.b else

6.b.2.b.1 if \( (\eta \leq e) \) then

   \( S = S \cup \text{QueryVPAC-tree}(T, \mathcal{C}_L, l_{\text{min}}, P_i, p_{\text{edeqv}}) \)

6.b.2.b.2 if \( (\eta \geq e) \) then

   \( S = S \cup \text{QueryVPAC-tree}(T, \mathcal{C}_R, l_{\text{min}}, P_i, p_{\text{edeqv}}) \)

endif

endif

7. return \( S \)

Let re-examine the example in Figure 2 and replace all notations \( \mathcal{C} \)’s as \( \mathcal{C} \)’s with vector \( E \) and flag \( G \) in each node. In \( N_1 \), we have \( e_1 = \delta(a, e) = \delta(a, j) = \delta(a, m) \) (i.e., \( E_1 = \{e, j, m\} \)); and, \( e \) and \( j \) are classified into its left sub-tree \( \mathcal{C}_L \), while \( m \) into \( \mathcal{C}_R \).

In \( N_2 \), we find that both \( e \) and \( j \) will be in its right sub-tree \( \mathcal{C}_R \) with \( G_L = 0 \) and \( G_R = 1 \) where \( G_L \) denotes \( G_L \) of \( N_2 \). Consider query \( q_i = j \). We search vpac-tree \( \mathcal{C} \) from the root node. In \( N_1 \), we find that \( \delta(a, q_i) = \delta(a, j) = e_1 \). Hence, \( edeqv = 1 \) and both \( N^2 \)
and $N^p$ need to be considered. In $N_2$, we obtain $p_{e|e|vq} = 1$ from its father node $N_1$ and find that $\eta_1 = \delta(h, q) = \sigma_2$. Now, we have $p_{e|e|vq} = 1$ and $G_{21} = 0$ so that $N_2$’s sub-tree $\mathcal{C}_{N_{21}}$ rooted at $N_3$ could be pruned away by a-cuts. Likewise, owing to $p_{e|e|vq} = 1$ and $G_{21} = 0$, $N_6$’s sub-tree $\mathcal{C}_{N_{61}}$ rooted at $N_{13}$ is also a-cuts. Therefore, the nodes need to be considered in level 3 of vpac-tree $N^p$ and $N^p$; while in vp-tree, we need to consider $N_1, N_5$ and $N_{10}$ in level 3.

4. Experimental Results and Discussions

To ease the discussion, the proposed schemes for coping with EMSMP using vp-tree and vpac-tree are referred to as vp and vpac, respectively. To evaluate their performances, we tested them by several datasets and compared their results against those by the m-BNDM algorithm. Their performances may be affected by $\sigma$ (the size of alphabet set $\Sigma$), $n$ (length of $T$), $r$ (number of patterns in $\mathcal{P}$) and $m_i$ (length of pattern $P_i$) for $1 \leq i \leq r$.

Our interest focuses on DNA alphabet for a long text and a large number of patterns so that we set $\sigma = 4$, $n = 1M$ and $r = 10K$. Regarding $m_i$, we generate five groups of lengths: $10\%$, $20\%$, $30\%$, $40\%$ and $50\%$. Note that the lengths of the first group are between 8 and 12, while those of the last one are between 40 and 60.

Our experimental platform is a personal computer with 3.1 GHz CPU (Intel i5-2400) and 32GB main memory. The operation system is CentOS 6.5 and all of the three algorithms are implemented in C language and then compiled by GNU Compiler Collection (gcc) 4.4.7 with optimization option -O3. The data in one test, including one text $T$ and five pattern sets/groups, are randomly generated from $\Sigma = \{a, g, c, t\}$. The averaged results from 100 independent tests would be reported in the following.

We summarize the total running time (in seconds) of the three algorithms for EMSMP under various pattern lengths in Table 1.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Pattern length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10±20%</td>
</tr>
<tr>
<td>m-BNDM</td>
<td>37.23</td>
</tr>
<tr>
<td>vp</td>
<td>1.67</td>
</tr>
<tr>
<td>vpac</td>
<td>1.47</td>
</tr>
</tbody>
</table>

From Table 1, we obtain two immediate findings:

(1) the execution times spent by the proposed schemes are better than that of m-BNDM, and
(2) vpac outperforms vp to a certain degree.

The superiority of vp and vpac over m-BNDM is evident in this experiment. Further, the alliance cuts are effective in vpac so that vpac is more efficient than vp.

Let us examine the behavior of vp and vpac. The tree construction time of vp and vpac is presented in Table 2. It is seen that the construction time needed by vpac-tree is slightly slower than that of vp-tree. The reason is that each node in vpac spends extra time to tackle set $E$ from its parent node, decide the flags $G_L$ and $G_R$ and pass set $E$ to child nodes in every node.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Pattern length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10±20%</td>
</tr>
<tr>
<td>vp</td>
<td>0.83</td>
</tr>
<tr>
<td>vpac</td>
<td>0.85</td>
</tr>
</tbody>
</table>

Table 3 shows the searching time of vp and vpac. Since vpac prunes away some unnecessary branches while searching by alliance cuts, the searching time of vpac is faster than that of vp.

Table 3 Searching time (in seconds) of vp and vpac

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Pattern length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10±20%</td>
</tr>
<tr>
<td>vp</td>
<td>0.84</td>
</tr>
<tr>
<td>vpac</td>
<td>0.61</td>
</tr>
</tbody>
</table>

From Tables 2 and 3, we realize when the length of the patterns becomes small, the tree construction time tends to be short. A small pattern length leads the number of data points with a same edit distance from the vantage point and that of the members of $V$ in each node increases so that the number of total nodes decreases and the height of the tree shrinks. Such a small pattern length also raises the chance for searching both sub-trees in a node. Thus, the searching time tends to be long. Meanwhile, the effect of alliance cuts in vp-tree goes vigorous. With the similar reasoning, a larger pattern length results in a larger total number of nodes, a relatively longer (or unshrunk) tree height, a shorter searching time and a weak effect of alliance cuts.

Let $N$ denote the number of total nodes in vp-tree and vpac-tree, $n_{vp}$ and $n_{vpac}$ be those of the searched nodes by vp and vpac, respectively. We could evaluate the searching effectiveness of vp and vpac by examining the search ratios defined as $\frac{n_{vp} + n_{vpac}}{N}$. (4)

To measure the effectiveness of vpac in cutting branches/nodes, we define the cut ratio as $\frac{(n_{vp} - n_{vpac})/n_{vp}}{n_{vp}}$. (5)

Table 4 reports $N$, $n_{vp}$, $n_{vpac}$, $n_{vp}/N$, $n_{vpac}/N$ and $(n_{vp} - n_{vpac})/n_{vp}$ in our experiment. Note that $N = 215985$ for pattern length of $10\%$ is very small (as compared to that of $20\%$). This is because $l_{min}$ is small in the case of $10\%$ and there are quite a lot identical sub-strings with length $l_{min}$ in $T$, which would be collected in vantage point set $V$ after the
vantage point is selected in each node. Therefore, $N$ becomes quite small.

Table 4 Results of $N$, $n_{vp}$, $n_{max}$ and cut-ratio

<table>
<thead>
<tr>
<th>Pattern length</th>
<th>$N$</th>
<th>$n_{vp}$</th>
<th>$n_{max}$</th>
<th>$n_{vp}/N$</th>
<th>$n_{max}/N$</th>
<th>$(n_{vp} - n_{max})/n_{vp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10±20%</td>
<td>20±20%</td>
<td>30±30%</td>
<td>40±40%</td>
<td>50±50%</td>
<td></td>
</tr>
<tr>
<td>$N'$</td>
<td>215985</td>
<td>999934</td>
<td>999977</td>
<td>999969</td>
<td>999961</td>
<td></td>
</tr>
<tr>
<td>$n_{vp}$</td>
<td>58973</td>
<td>50662</td>
<td>33499</td>
<td>26011</td>
<td>21421</td>
<td></td>
</tr>
<tr>
<td>$n_{max}$</td>
<td>40789</td>
<td>41444</td>
<td>27608</td>
<td>21598</td>
<td>17896</td>
<td></td>
</tr>
<tr>
<td>$n_{vp}/N$</td>
<td>27.30%</td>
<td>5.07%</td>
<td>3.35%</td>
<td>2.60%</td>
<td>2.14%</td>
<td></td>
</tr>
<tr>
<td>$n_{max}/N$</td>
<td>18.88%</td>
<td>4.11%</td>
<td>2.76%</td>
<td>2.16%</td>
<td>1.79%</td>
<td></td>
</tr>
<tr>
<td>$(n_{vp} - n_{max})/n_{vp}$</td>
<td>30.83%</td>
<td>18.79%</td>
<td>17.58%</td>
<td>16.96%</td>
<td>16.45%</td>
<td></td>
</tr>
</tbody>
</table>

It is seen from Table 4 that the search ratio tends to decrease when pattern length increases. As we mentioned before, when the pattern length is small, the data points (corresponding to $T$) with a same edit distance from the vantage point grow larger. Those with the same distance as median $e$ would be decomposed into both sub-trees of the node. Consequently, both sub-trees have to be searched, when the edit distance between the query point and the vantage point is the same as $e$ in a node. When pattern length is 10±20%, the search ratios are 27.30% and 18.88% for $vp$ and $vpac$, respectively. On the other hand, when the pattern length is 50±20%, the search-ratio is only 2.14% (1.79%) for $vp$ ($vpac$).

Regarding the cut ratio, it tends to decrease when pattern length increases. The reason is that when pattern length grows, the range of edit distances becomes large. Then the chance that the edit distance between a vantage point and a query point is the same as median $e$ is declined. Therefore, the chance of searching both sub-trees is also reduced. In the case of 10±20%, the cut ratio is 30.83%; while in the case of 50±20%, it reduces to 16.45%.

The aforementioned outcomes and discussion explain the effectiveness of $vpac$ in pruning away unnecessary branches, which makes $vpac$ more efficient than $vp$.

5. Concluding Remarks

By mapping the sub-strings in $T$ and prefixes of all $P_i$’s with length $l_{max}$, as data points in the edit distance-based metric space for 1≤$i$≤$r$, we design two schemes which construct the data points of $T$ into $vp$-tree and $vpace$-tree, respectively, and search data points of $P_i$ in the trees, for solving EMSMP. The expected tree constructing time is $O(l_{max}/\omega \times l_{max} \times n \log n)$ and the expected searching time is $O(l_{max}/\omega \times l_{max} \times n \log n)$ for either $vp$-tree or $vp$-ac-tree.

The experimental results show that the proposed schemes using $vp$-tree and $vpace$-tree are more efficient than the m-BNDM algorithm. In addition, the effectiveness of alliance cuts in $vpac$-tree is significant, especially when the pattern length is small.

In the near future, we would like to test more data sets with other alphabets (than DNA) to see whether the benefits obtained so far could still retain. Furthermore, we would like to cope with the approximate multiple string matching problem using the ideas of $vp$-tree and $vpace$-tree.

References


Appendix A

The construction of $vp$-tree with respect to $X = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\}$, as shown in Figure 2(a) is illustrated level by level as follows. In the
beginning, we choose a as the first vantage point and we have $X = X(a)$ and $V_1 = \{a\}$. Among all $\delta(a, x)$’s, the median distance is $e_1 = \delta(a, j) = \delta(a, e) = \delta(a, m)$ as shown in Figure A(a) where e, j and m are denoted as red triangles. Based on median $e_1$, we apply the revised refined ball partition procedure to obtain $X'_i = \{i, k, b, d, h, e, j\}$ and $X'_s = \{m, l, c, n, f, o, g\}$. The node with respect to $X$ would be $N^1 = \{a\}, e_1$, $\mathbb{C}^e_i$, $\mathbb{C}^e_s$ where $\mathbb{C}^e_i$ and $\mathbb{C}^e_s$ are the sub-trees with respect to $X'_i$ and $X'_s$, respectively, after they are recursively built as shown in Figure A(b). Let us observe the left child node $N^2$ of $N^1$. In node $N^2$, $X^2 = \{i, k, b, d, h, e, j\}$ and median $e_2 = \delta(b, d) = \delta(b, j)$. Then, we obtain $X'_i = \{h, i, d\}$ and $X'_s = \{j, e, k\}$ as shown in Figure A(c) and (d). The left child node $N^3$ of $N^2$ chooses $d$ to be its vantage point where $X^3 = X^2 \setminus \{d\}$, $V_2 = \{b\}$ and median $e_3 = \delta(d, h)$. Hence, $X'_i = \{h\}$ and $X'_s = \{i\}$ as shown in Figure A(e). In node $N^4 (N^3)$ with respect to $X^4 = X^3$ ($X^3 = X'_s$), $h$ (i) is the only point and the vantage point so that it is the leaf node without any sub-tree (see Figure A(f)). The returned pointers $\mathbb{C}^e_i$ and $\mathbb{C}^e_s$, after the construction of $N^3$ and $N^4$, respectively, cause $N^3$ to be physically built. For $N^4$, vantage point is $e$ and $e_6 = \delta(e, j)$ (Figure A(g)), which makes nodes $N^5$ and $N^6$ (containing j and k with pointers $\mathbb{C}^e_j$ and $\mathbb{C}^e_k$ respectively) be built as leaf nodes. $\mathbb{C}^e_i$ and $\mathbb{C}^e_k$ cause $N^5$ to be built with $\mathbb{C}^e_s$; and consequently, $\mathbb{C}^e_i$ and $\mathbb{C}^e_j$ (to $N^6$ and $N^7$, respectively) make $N^7$ be built with pointer $\mathbb{C}^e_k$ as shown in Figure A(h).

The sub-tree pointed by $\mathbb{C}^e_i$, is constructed by the same recursive approach. Figure A(i) and (j) depict that $c$ is the vantage point of $N^8$ with $e_9 = \delta(c, m)$, $X'_i = \{f, l, m\}$ and $X'_s = \{g, n, o\}$. Figure A (k) show vantage point $f$ (l, m) in $N^{10} (N^{11}, N^{12})$ where $N^{11}$ and $N^{12}$ become leaf nodes and (l) illustrates the partially constructed vp-tree. Figure A(m) gives vantage point g in $N^{13}$ and the completion of $N^{11}$ and $N^{12}$; and A(n) presents the final vp-tree.
Figure A. The constructing vp-tree for Example 1