Exact Multiple String Matching Problem for DNA Alphabet

Yi-Kung Shieh, Shyong Jian Shyu and Richard Chia-Tung Lee
Department of Computer Science
Nation 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 $T = t_1t_2 \ldots t_n$ 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 $1 \le i \le r$. We regard all substrings 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) $\mathcal{T}$. Then, EMSMP can be resolved by searching all points of P in $\mathcal{T}$. We further enhance $\mathcal{T}$ into vpac-tree $\mathcal{C}$ (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 Σ consisting of σ symbols and *strings* whose constituent members are from the symbols of Σ. Let $T = t_1t_2 \ldots t_n$ be a text string and $P = \{P_1, P_2, \ldots, P_r\}$ be a set of patterns where $P_i = p_1^i p_2^i \ldots p_{m_i}^i$, m_i (= $||P_i||$) is the length of P_i and t_j, $p_s^i \in \Sigma$ for $1 \le j \le n$, $1 \le i \le r$ and $1 \le s \le m_i$. Let $T[i, j]$ be the sub-string staring at i and ending at j of T (i.e., $T[i, j] = t_it_{i+1} \ldots t_{j-1}t_j$) where $1 \le i \le j \le n$. The *exact multiple string matching problem* (EMSMP) is to find all ending positions λ_i's in T such that $T[\lambda_i - m_i + 1, \lambda_i] = P_i$ (i.e., $t_{\lambda_i - m_i + 1}t_{\lambda_i - m_i + 2} \ldots t_{\lambda_i} = p_1^i p_2^i \ldots p_{m_i}^i$) for $1 \le i \le r$. Assume that T = "thoseeasycasesmaynotbeeasy" and P = {"those", "easy", "test", "se"}. The answer should be {{5}, {9,26}, ∅, {5,13}} because P_1 = "those" = $t_1t_2t_3t_4t_5$, P_2 = "easy" = $t_6t_7t_8t_9$ = $t_{23}t_{24}t_{25}t_{26}$ and P_4 = "se" = t_4t_5 = $t_{12}t_{13}$. 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 $l_{\min}$, among the patterns in P and set a window W with length $l_{\min}$ 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., $P_i[l_{\min}+1, m_i]$) 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 $\mathcal{T}$ and those in P are regarded as queries. EMSMP can thus be solved by searching all points of P in $\mathcal{T}$. The second scheme further explores the possibility to cut more unnecessary branches in $\mathcal{T}$ 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 simulate 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 $l_{\min}$ among all patterns. Let the prefix with length $l_{\min}$ of P_i be P'_i. It reverses all P'_i's which are denoted as P_i^R's, and then concatenates them to a long pattern P' (i.e.: $P' = P_1^R P_2^R \ldots P_r^R$). An auxiliary 0/1 $\sigma \times \lceil \|P'\|/\omega \rceil$ storage B is used to record the positions of all characters of Σ in P'. Specifically, if the character of i^{th} position in P' is x, the i^{th} bit of $B[x]$ is 1; otherwise 0. Note that we use $\lceil \|P'\|/\omega \rceil$'s computer words to record $B[x]$, where ω 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 $W = T[i, i+l_{\min}-1]$ and $W = T[1, l_{\min}]$ 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 $\lceil \|P'\|/\omega \rceil$ 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 $last = l_{\min}$. 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 $DF = (10^{l_{\min}-1})^r$. If $D'\&DF \neq 0$, there is a suffix with length $l_{\min}-j+1$ 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 $l_{\min}$. This applies that the string in W is equal the suffix of some pattern(s). By checking whether $D'\&0^{\|P\|-l_{\min}\times i}10^{l_{\min}-1}0^{l_{\min}\times(i-1)}$ is zero for $1 \le i \le r$, we know the prefix belongs to P_i; further checking whether $P_i[l_{\min}, m_i] = T[i+l_{\min}, i+l_{\min}-1]$ lead us to report an exact match with ending position $i+l_{\min}-1$. If $D'\&DF \neq 0$ and $j \neq 1$, *last* is updated to be $j-1$. After testing $D'\&DF$, we set $D = (D'<<1)\&(1^{l_{\min}-1}0)^r$ where << is the left shift operation. If $D \neq 0$, the character of $(j-1)^{\text{th}}$ is read and it continuously utilizes the above method. Otherwise, in order to align the suffix (with length $l_{\min}-last$) 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) $d(x, y) \le d(x, z) + d(z, y)$ (triangle inequality).

The vp-tree, initially invented by Yianilos [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 γ 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_j's such that the distance between q and x_i is no more than γ (i.e., $d(q, x_j) \le \gamma$) for $x_j \in X$. Once vp-tree $\mathcal{T}$ with respect to X is built, the query specified by (q, γ) could be easily answered by searching points in $\mathcal{T}$ whose distance from q is within the range γ.

The construction of vp-tree $\mathcal{T}$ with respect to X is a recursive process involving a *ball partition* procedure. This procedure starts from choosing a point, say x_v, which is called the *vantage point*, randomly from X. All distances between x_v and those in $X\backslash\{x_v\}$ (i.e., $d(x_v, x_j)$'s for $x_j \in X\backslash\{x_v\}$) are computed. Let e be the median of all $d(x_v, x_j)$'s. Then, $X\backslash\{x_v\}$ would be evenly partitioned into X_L and X_R such that

$$X_L = \{\, x_l \mid d(x_v, x_l) \le e \text{ for } x_l \in X\backslash\{x_v\}\} \text{ and}$$
$$X_R = \{\, x_r \mid d(x_v, x_r) \ge e \text{ for } x_r \in X\backslash\{x_v\}\}. \quad (1)$$

Conceptually, such data decomposition can be regarded as the partitioning result of *ball* $B(x_v, e)$ centered at vantage point x_v with radius e such that X_L and X_R locate inside and outside B, respectively. That's why we call it the ball partition. After X_L and X_R are recursively constructed as left and right sub-trees $\mathcal{T}_L$ and $\mathcal{T}_R$, respectively. We could build a node N consisting of $(x_v, e, \mathcal{T}_L, \mathcal{T}_R)$ to be the root of the vp-tree $\mathcal{T}$ 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, $\mathcal{T}$ is a balanced binary tree.

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

distances from q are no greater than γ would be collected and reported as the final solution set. The simple computation of η and the possibility of pruning away one sub-tree at each node give rise to an optimism for efficiently answering the range query problem by searching in the vp-tree.

Our designs of applying vp-tree and its refined version with additional branch-cutting capability to resolve EMSMP are proposed in the following section.

3. Peoposed Schemes for Solving EMSMP

Our strategy for coping with EMSMP is constructing the substrings in text T as vp-tree $\mathcal{T}$ and then reporting the matches of patterns in P by searching them in $\mathcal{T}$. 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 $\|t\| = \|p\|$. 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 $\mathcal{T}$ in Section 3.1. The algorithm for searching P_i in $\mathcal{T}$ for $1 \le i \le r$ is designed in Section 3.2. A brilliant refinement on the vp-tree with additional capabilities to cut off impossible sub-trees more efficiently while searching is explained in Section 3.3. The corresponding searching algorithm is presented in Section 3.4.

3.1 Basic vp-tree

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

For each pair of x_j and q_i, if $\delta(q_i, x_j) \ne 0$ (or $x_j \ne q_i$), 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, $1 \le j \le n-l_{\min}+1$ and $1 \le i \le r$,
(a) if $\delta(q_i, x_j) \ne 0$, $P_i \ne T[j, j+m_i-1]$;
(b) $P_i = T[j, j+m_i-1]$, only if $\delta(q_i, x_j) = 0$.

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

To construct vp-tree $\mathcal{T}$ for $X = \{x_1, x_2, \ldots, x_{n-l_{\min}+1}\}$, we perform a *refined ball partition* procedure first. A data point, namely x_v, is randomly selected from X to be the vantage point. We delete x_v from X and initialize a set V to hold x_v (i.e., $X = X \backslash \{x_v\}$ and $V = \{x_v\}$). All edit distances between x_v and x_j's ($\delta(x_v, x_j)$'s) are computed for $x_j \in X$ and once we find $\delta(x_v, x_j) = 0$, which means x_j is the same as the vantage point x_v (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 $\delta(x_v, x_j)$'s. We partition X evenly into two subsets X_L and X_R such that

$$X_L = \{a \mid \delta(x_v, a) \le e \text{ and } a \in X\} \text{ and}$$
$$X_R = \{b \mid \delta(x_v, b) \ge e \text{ and } b \in X\} \quad (2)$$

where $|X_L| - |X_R| \le 1$.

After the refined ball partition, we recursively build sub-trees $\mathcal{T}_L$ and $\mathcal{T}_R$ 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 $\mathcal{T}_L$ and $\mathcal{T}_R$ are built, we create a node N consisting of V, e, $\mathcal{T}_L$ and $\mathcal{T}_R$ as the root node of $\mathcal{T}$. The basic structure of node N in $\mathcal{T}$ is illustrated in Figure 1. Note that, without ambiguity, we refer to $\mathcal{T}$ ($\mathcal{T}_L$ or $\mathcal{T}_R$) as a tree (sub-tree) or its corresponding pointer in physical implementation interchangeably, and N (N_L 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).

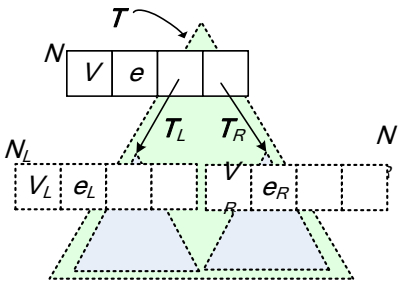


Figure 1. Structure of node N and its sub-trees in our vp-tree for EMSMP

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 $l_{\min}$ is in time $O(\lceil l_{\min}/\omega \rceil \times l_{\min})$ by applying the bit-vector algorithm [4], were ω 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 kth selection problem) or the bucket-sort. The height of vp-tree is $O(\log_2 n)$ 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 $O(\lceil l_{\min}/\omega \rceil \times l_{\min} \times n \log_2 n)$.

Algorithm 1. Constructing a vp-tree

Input: Data point set $X = \{x_1, x_2, \ldots, x_{n-l\min+1}\}$ corresponding to text $T = t_1 t_2 \ldots t_n$

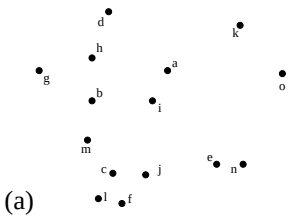
Output: Vp-tree $\mathcal{T}$ with respect to X (T)

RefinedBallPartition(X, x_v)

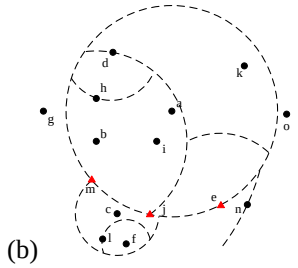
1. $X = X \backslash \{x_v\}$ and $V = \{x_v\}$
2. **for** (each $x_j \in X$) **do**
   Calculate $\delta(x_v, x_j)$
   if ($\delta(x_v, x_j) = 0$) **then**
   $X = X \backslash \{x_j\}$ and $V = V \cup \{x_j\}$
   endfor
3. e = median of $\{\delta(x_j, x_v) \mid x_j \in X\}$
4. Evenly divide X into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

ConstructVP-tree(X)

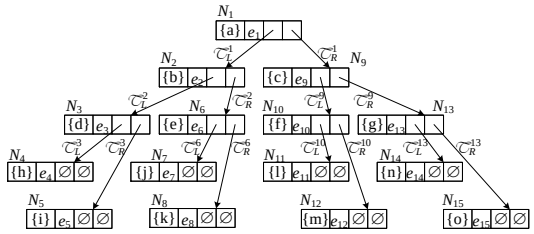
1. **if** ($X = \varnothing$) **then return** an empty vp-tree
2. Arbitrarily choose a point from X, say x_v, to be the vantage point
3. (V, e, X_L, X_R) = *RefinedBallPartition*(X, x_v)
4. $\mathcal{T}_L$ = *ConstructVP-tree*(X_L)
5. $\mathcal{T}_R$ = *ConstructVP-tree*(X_R)
6. Create node N = (V, e,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



(a)



(b)



(c)

Figure 2. Vp-tree $\mathcal{T}$ in Example 1

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 $e_1 = \delta(\text{a, e}) = \delta(\text{a, j}) = \delta(\text{a, m})$; 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 $1 \le i \le r$, receives matches in T, we regard $P_i[1, l_{\min}]$, the prefix with length $l_{\min}$ 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 γ (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 $\eta = \delta(x_v, q_i)$ for a certain $x_v \in V$. On condition that $\eta = 0$ (that is, $x_v = T[v, v+l_{\min}-1] = P_i[1, l_{\min}] = q_i$), we realize that all the points in V are solution candidates by Property 1 (b). A further tail checking procedure for each candidate x_v ($\in V$) goes on: If $P_i[l_{\min}+1, m_i] = T[v+l_{\min}, v+m_i-1]$, position $v+m_i-1$ of T is in the solution set; otherwise, ignore x_v. When all x_v'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 η with e. If $\eta > e$, left sub-tree $\mathcal{T}_L$ could be cut off; while $\eta < e$, right sub-tree $\mathcal{T}_R$ could be pruned away. We call this a *b-cut* owing to the merit of binary search. Property 2 emphasizes this.

Property 2. If $\eta\ (=\delta(x_v, q_i)) > e$ in $N = (V, e, \mathcal{T}_L, \mathcal{T}_R)$, $\mathcal{T}_L$ can be b-cut; and if $\eta < e$, $\mathcal{T}_R$ can be b-cut.

In short, if $\delta(x_v, q_i) \le e$, the left sub-tree $\mathcal{T}_L$ will be searched; in addition, if $\delta(x_v, q_i) \ge e$, the right sub-tree $\mathcal{T}_R$ 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 $P_1, P_2, \ldots, P_r$ in P in text T via searching q_i in $\mathcal{T}$ is shown in Algorithm 2 where $1 \le i \le r$. Because the height of vp-tree is $O(\log_2 n)$, the excepted time complexity of Algorithm 2 is $O(\lceil l_{\min}/\omega \rceil \times l_{\min} \times \log_2 n)$.

Algorithm 2. Searching patterns in a vp-tree

Input: Text $T = t_1 t_2 \ldots t_n$, $\mathcal{T}$ with respect to X (T), $l_{\min}$ and pattern set $P = \{P_1, P_2, \ldots, P_r\}$

Output: $S = \{S_1, S_2, \ldots S_r\}$ in which S_i consists of the ending positions of the substrings in T which are exactly equal to P_i for $1 \le i \le r$

QueryVP(T, $\mathcal{T}$, $l_{\min}$, P_i)

1. **if** ($\mathcal{T} = \varnothing$) **then return** $\varnothing$
2. Obtain (V, e, $\mathcal{T}_L$, $\mathcal{T}_R$) from the node pointed by $\mathcal{T}$
3. Let $q_i = P_i[1, l_{\min}]$
4. Choose any point, say x_v, in V as the vantage point
5. Compute $\eta = \delta(x_v, q_i)$ and set $S = \varnothing$

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

6.a.1 **for** (each $x_v \in V$) **do** // tail checking
if ($T[v+l_{\min}, v+m_i-1] = P_i[l_{\min}+1, m_i]$) **then**
$S = S \cup \{v+m_i-1\}$
endif
endfor

6.b **else**

6.b.1 **if** ($\delta(x_v, q_i) \le e$) **then**
$S = S \cup QueryVP(T, \mathcal{T}_L, l_{min}, P_i)$

6.b.2 **if** ($\delta(x_v, q_i) \ge e$) **then**
$S = S \cup QueryVP(T, \mathcal{T}_R, l_{min}, P_i)$
endif
endif

7. **return** S

SolveEMSMP(T, P)

1. $X = \{x_1, x_2, \ldots, x_{n+l_{\min}-1}\}$ where $x_j = T[j, j+l_{\min}-1]$ for $1 \le j \le n+l_{\min}-1$
2. $\mathcal{T} = ConstructVP\text{-}tree(X)$
3. $l_{\min} = \min\{\|P_i\| \mid P_i \in P\}$
4. **for** (each i, $1 \le i \le r$) **do**
$S_i = QueryVP(T, \mathcal{T}, l_{\min}, P_i)$
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 $\delta(x_v, q_i) = 0$ occurs for some $x_v \in V$ 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 $\mathcal{T}$ is shown in Figure 2, consider a query q_i that is exactly equal to point j (i.e., q_i = j). We search $\mathcal{T}$ and start from root N_1. In N_1 (= ({a}, e^1, $\mathcal{T}_L^1$, $\mathcal{T}_R^1$)), δ(a, q_i) = δ(a, j) $\neq 0$ and δ(a, q_i) = e_1 where e_1 is the median distance of N_1. Hence, we have to search both $\mathcal{T}_L^1$ and $\mathcal{T}_R^1$ (rooted at N_2 and N_9, respectively). In N_2, because δ(b, q_i) $\neq 0$ and δ(b, q_i) = e_2, both $\mathcal{T}_L^2$ and $\mathcal{T}_R^2$ need to be considered. In N_3, since δ(d, q_i) $\neq 0$ and δ(d, q_i) > e_3, only N_5 (but not N_4 by Property 2) would be searched. In leaf N_5, we find that δ(i, q_i) $\neq 0$ and no more child node would be searched. Hence, there is no solution in sub-tree $\mathcal{T}_L^2$. Let us consider $\mathcal{T}_R^2$. In N_6. owing to δ(e, q_i) $\neq 0$ and δ(e, q_i) = e_3, N_7 and N_8 need to be considered. In leaf N_7, we find that δ(j, q_i) = 0 so that S = {j} and stop searching any descendant node of N_7. In leaf N_8, δ(j, q_i) $\neq 0$ and no child node would be searched. By now, the whole left sub-tree $\mathcal{T}_L^1$ has been searched and we go on searching $\mathcal{T}_R^1$ rooted at N_9. In N_9, δ(c, q_i) $\neq 0$ and δ(c, q_i) < e_9 so that only sub-tree $\mathcal{T}_L^9$ (but not $\mathcal{T}_R^9$ by Property 2) need to be considered. In N_{10}, since δ(f, q_i) $\neq 0$ and δ(f, q_i) > e_{10}, only $\mathcal{T}_L^{10}$ (instead of $\mathcal{T}_R^{10}$) need to be considered. In leaf N_{12}, δ(m, q_i) $\neq 0$ and no more child node could be searched. Finally, we obtain S_i = {j}. Note that in this case ten out of the fifteen nodes are searched.

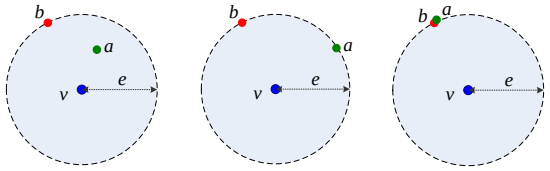
3.3 Vp-tree with alliance cut

When querying q_i in $\mathcal{T}$ rooted at N, we may encounter a situation that both $\mathcal{T}_L$ and $\mathcal{T}_R$ 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 δ(a, q_i) = e_1, δ(b, q_i) = e_2 and δ(e, q_i) = e_6, respectively. Concerning such situation ($\delta(x_v, q_i) = e$ in N consisting V and e where $x_v \in V$), we shall introduce a skill, referred to as the *alliance cut*, to determine whether N_L's (or N_R'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 $\delta(v, a) = \delta(v, b)$.



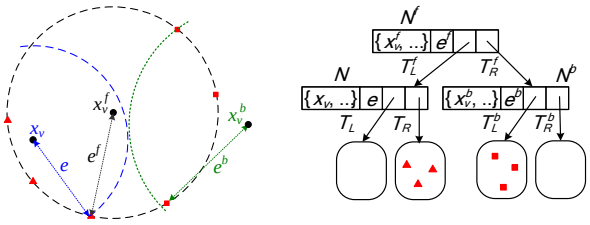
(a) (b) (c)
Figure 3. Some examples for Property 3

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_1 with vantage point a and medium distance e_1, we find thee points e, j and m such that δ(a, e) = δ(a, j) = δ(a, m) = e_1. By Algorithm 2, we know that both sub-trees $\mathcal{T}_L^1$ and $\mathcal{T}_R^1$ 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_1. Observing $\mathcal{T}_L^1$ rooted at N_2 in Figure 2 (b), we know that N_2 classifies e and j into $\mathcal{T}_R^2$. It means that there is no need to search $\mathcal{T}_L^2$ (in which no solution candidate exists). In addition, observing $\mathcal{T}_R^1$ rooted at N_9, we find that m is classified into $\mathcal{T}_L^9$. It means that there is no need to search $\mathcal{T}_R^9$ (since there is no solution candidate). Algorithm 2 could detect the latter (no need to search $\mathcal{T}_R^9$) by b-cut at N_9, but it could not explore the former (so that $\mathcal{T}_L^2$ would be searched in Algorithm 2). Our a-cut aims at cutting off such $\mathcal{T}_L^2$ (and $\mathcal{T}_R^9$).

Let N denote a certain node in $\mathcal{T}$, N^f be its father, N^b be its brother, $\mathcal{T}_L$ and $\mathcal{T}_R$ be its two sub-trees rooted at N_L and N_R where $N = (V, e, \mathcal{T}_L, \mathcal{T}_R)$, $N^f = (V^f, e^f, \mathcal{T}_L^f, \mathcal{T}_R^f)$ and $N^b = (V^b, e^b, \mathcal{T}_L^b, \mathcal{T}_R^b)$. The alliance cut decision in N (N^b) about cutting $\mathcal{T}_L$ or $\mathcal{T}_R$ ($\mathcal{T}_L^b$ or $\mathcal{T}_R^b$) relies on N^f's detection about $E^f \neq \varnothing$ where $E^f = \{ x_v^f \mid \delta(x_v^f, q_i) = e^f \text{ for } x_v^f \in X^f \}$. We call it an alliance cut for the reason that father's (N^f's) knowledge is alliance with its son N (N^b) and after N (N^b) finishes the revised ball partition, such knowledge is capable of judging whether some sub-tree(s) $\mathcal{T}_L$ or $\mathcal{T}_R$ ($\mathcal{T}_L^b$, $\mathcal{T}_R^b$) may be cut off.

Property 4. If $\delta(x_v^f, q_i) = e^f$ and $E^f \neq \varnothing$ in $N^f = (V^f, e^f, \mathcal{T}_L, \mathcal{T}_R)$ with respect to X^f, $\mathcal{T}_L$ or $\mathcal{T}_R$ ($\mathcal{T}_L^b$, $\mathcal{T}_R^b$) in N (N^b) which contains no member of E^f can be a-cut by the alliance of (x_v^f, x_v) where $E^f = \{ x_v^f \mid \delta(x_v^f, q_i) = e^f \text{ for } x_v^f \in X^f \}$.

Figure 4 gives a general case that a-cuts do occur where (a) depicts a situation that six points x_j's (in red) are detected with $\delta(x_v^f, x_j) = e^f$ (i.e. $|E^f| = 6$) in N^f and (b) tells that three of the six are decomposed into $\mathcal{T}_R$ in N (in red triangles) and the other three into $\mathcal{T}_L^b$ in N^b. Then, $\mathcal{T}_L$ and $\mathcal{T}_R^b$, which contain no member of E^f, can be pruned by a-cuts in N from the alliance of vantage points x_v and x_v^f.



(a) (b)
Figure 4. Example of alliance cuts in vp-tree

The alliance cut is considered at N (N^b) only when $E^f \neq \varnothing$ in its father N^f. On condition that $E^f = \varnothing$ in N^f, we simply follow the rules of searching descendants in Algorithm 2 where b-cuts may happen.

To fulfill the idea of alliance cut, we enhance our vp-tree and refer to the refined result as the *vp-tree with alliance cut capability* (vpac-tree for short), denoted as $\mathcal{T}$. The construction of $\mathcal{T}$ has the similar structure as that of T. Additionally, we maintain an *equal-distance-to-e* set E in each node (consisting of V and e) with respect to X such that

$$E = \{ x_j \mid \delta(x_v, x_j) = e \text{ for some } x_v \in V \text{ and all } x_j\text{'s} \in X \} \quad (3)$$

to include those x_j's($\in X$) whose $\delta(x_v, x_j) = e$.

Consider again node N, its father N^f, brother N^b, two sub-trees $\mathcal{T}_L$ and $\mathcal{T}_R$. N^f with data point set X^f has its own E^f vector, which would be decomposed as $E_L{}^f$ and E_R^f according to X_L^f and X_R^f, respectively. During the construction of N, N inherits X (either X_L^f or X_R^f,) and E' (either E_L^f or E_R^f, respectively) from N_f. N performs the refined ball partition procedure to obtain (V, e, X_L, X_R) and decides the *go-on* flags by:

$$G_L = \begin{cases} 1 & \text{if } x_a \in E' \text{ for any } x_a \in X_L; \\ 0 & \text{otherwise,} \end{cases} \quad \text{and}$$

$$G_R = \begin{cases} 1 & \text{if } x_b \in E' \text{ for any } x_b \in X_R; \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

That is, $G_L = 1$ means sub-tree $\mathcal{T}_L$ contains at least one data point, say x_a, satisfying $\delta(x_v^f, x_a) = e^f$ in N^f (or $x_a \in E^f$); otherwise, $\mathcal{T}_L$ does not contain any member in E^f. Thus, $\mathcal{T}_L$ could be a-cut when $\delta(x_v^f, q_i) = e^f$ and $G_L = 0$ occur for some q_i by Properties 3 and 4. The same reasoning applies for G_R. Surely, N establishes its own E with respect to X using Eq. (3). It would be further decomposed into E_L and E_R according to X_L and X_R. Recursively, $\mathcal{T}_L$ ($\mathcal{T}_R$) is built with respect to X_L and E_L (X_R and E_R). The recursion

returns an empty node on condition that the input data point set is empty. After the construction of $\mathcal{T}_L$ and $\mathcal{T}_R$, we establish $N = (V, e, \mathcal{T}_L, \mathcal{T}_R, (G_L, G_R))$ as the root of $\mathcal{T}$. 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 $X = \{x_1, x_2, \ldots, x_{n-l_{\min}+1}\}$ corresponding to text $T = t_1t_2 \ldots t_n$ and *equal-distance-to-e* set E' (initially $E' = \varnothing$)

Output: *Vpac-tree* $\mathcal{T}$ *of* X (T)

ConstructVPAC-tree(X, E')

1. **if** ($X = \varnothing$) **then return** an empty vp-tree
2. Arbitrarily choose a point from X, say x_v, to be the vantage point
3. $(V, e, X_L, X_R) = RefinedBallPartition(X, x_v)$
4. $G_L = 0$ and $G_R = 0$
5. **if** ($x_a \in E'$ for any $x_a \in X_L$) **then** $G_L = 1$
6. **if** ($x_b \in E'$ for any $x_b \in X_R$) **then** $G_R = 1$
7. $E_L = \varnothing$ and $E_R = \varnothing$
8. **for** (each a, $x_a \in X_L$) **do**
   if ($\delta(x_v, x_a) = e$) **then** $E_L = E_L \cup \{x_a\}$
9. **for** (each b, $x_b \in X_R$) **do**
   if ($\delta(x_v, x_b) = e$) **then** $E_R = E_R \cup \{x_b\}$
10. $\mathcal{T}_L = ConstructVPAC\text{-}tree(X_L, E_L)$
11. $\mathcal{T}_R = ConstructVPAC\text{-}tree(X_R, E_R)$
12. Create a node $N = (V, e, \mathcal{T}_L, \mathcal{T}_R, G)$ with pointer $\mathcal{T}$ where $G = (G_L, G_R)$
13. **return** $\mathcal{T}$

3.4 Searching in basic vpdc-tree

When dealing with a query point q_i (= $P_i[1, l_{\min}]$) at node N containing V and e in $\mathcal{T}$, we simply compute $\eta = \delta(x_v, q_i)$ where $1 \le i \le r$ and $x_v \in V$. If $\eta = 0$, a subsequent tail checking is performed for each members 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 *equal-distance-to-e-between-query-vantange* is set, i.e., *edeqv* = 1; otherwise *edeqv* = 0. This flag would be passed to N's sub-trees $\mathcal{T}_L$ and $\mathcal{T}_R$ rooted at N_L and N_R which rename it as "*p_edeqv*". Of course, N receives such *p_edeqv* from its father N^f. Note that in the root node, *p_edeqv* = 0 since it has no parent node.

On condition that N gets *p_edeqv* = 1 from its father N^f, $G = (G_L, G_R)$ becomes useful. If G_L (G_R) = 0, $\mathcal{T}_L$ ($\mathcal{T}_R$) can be a-cut directly by Property 4. In fact, $\mathcal{T}_L$ ($\mathcal{T}_R$) would only be searched when $\eta \le e$ (it is not b-cut by vantage point x_v in N) and $G_L = 1$ (it cannot be a-cut by the alliance of x_v in N and x_v^f in N^f). Otherwise (*p_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 $1 \le i \le r$.

Algorithm 4. Searching a pattern in a vpac-tree

Input: Text T, vpac-tree $\mathcal{T}$ of T, length $l_{\min}$, pattern P_i and flag *p_edeqv* (initially, *p_edeqv* = 0)

Output: S consisting of the ending positions of the substrings in T which are exactly equal to P_i

QueryVPAC-tree(T, $\mathcal{T}$, $l_{\min}$, P_i, *p_edeqv*)

1. **if** ($\mathcal{T} = \varnothing$) **then return** $\varnothing$
2. Obtain $(V, e, \mathcal{T}_L, \mathcal{T}_R, (G_L, G_R))$ in the node pointed by $\mathcal{T}$
3. Let $q_i = P_i[1, l_{\min}]$
4. Choose any point, say x_v, in V as the vantage point
5. Compute $\eta = \delta(x_v, q_i)$ and set $S = \varnothing$

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

6.a.1 **for** (each $x_v \in V$) **do** // tail checking
   if ($T[v+l_{\min}, v+m_i-1] = P_i[l_{\min}+1, m_i]$) **then** $S = S \cup \{v+m_i-1\}$
   endfor

6.b **else**

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

6.b.2.a **if** (*p_edeqv* = 1) **then**

6.b.2.a.1 **if** ($\eta \le e$ **and** $G_L = 1$) **then** $S = S \cup QueryVPAC\text{-}tree(T, \mathcal{T}_L, l_{\min}, P_i, edeqv)$

6.b.2.a.2 **if** ($\eta \ge e$ **and** $G_R = 1$) **then** $S = S \cup QueryVPAC\text{-}tree(T, \mathcal{T}_R, l_{\min}, P_i, edeqv)$
   endif

6.b.2.b **else**

6.b.2.b.1 **if** ($\eta \le e$) **then** $S = S \cup QueryVPAC\text{-}tree(T, \mathcal{T}_L, l_{\min}, P_i, edeqv)$

6.b.2.b.2 **if** ($\eta \ge e$) **then** $S = S \cup QueryVPAC\text{-}tree(T, \mathcal{T}_R, l_{\min}, P_i, edeqv)$
   endif
   endif
   endif

7. **return** S

Let re-examine the example in Figure 2 and replace all notations $\mathcal{T}$s as $\mathcal{T}$'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 $\mathcal{T}_L^1$, while m into $\mathcal{T}_R^1$. In N_2, we find that both e and j will be in its right sub-tree $\mathcal{T}_R^2$ with $G_L^2 = 0$ and $G_R^2 = 1$ where G_L^2 denotes G_L of N_2. Consider query q_i = j. We search vpac-tree $\mathcal{T}$ 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^9 need to be considered. In N_2, we obtain $p\_edeqv = 1$ from its father node N_1 and find that $\eta_2 = \delta(b, q_i) = e_2$. Now, we have $p\_edeqv = 1$ and $G_L^2 = 0$ so that N_2's sub-tree $\mathcal{T}_L^2$ rooted at N_3 could be pruned away by a-cut. Likewise, owing to $p\_edeqv = 1$ and $G_L^9 = 0$, N_9's sub-tree $\mathcal{T}_R^9$ rooted at N_{13} is also a-cut. Therefore, the nodes need to be considered in level 3 of vpac-tree N^6 and N^{10}; while in vp-tree, we need to consider N_3, N_6 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 σ (the size of alphabet set Σ), n (length of T), r (number of patterns in P) and m_i (length of pattern P_i) for $1 \le i \le r$.

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%, 20±20%, 30±20%, 40±20% and 50±20%. 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 Σ = {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 1 Results of total running time (in seconds) for m-BNDM, vp and vpac

Algorithm	Pattern length				
	10±20%	20±20%	30±20%	40±20%	50±20%
m-BNDM	37.23	16.70	14.50	13.84	13.43
vp	1.67	2.81	3.31	3.94	4.69
vpac	1.47	2.74	3.26	3.90	4.63

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 2 Tree construction time (in seconds) of vp and vpac

Algorithm	Pattern length				
	10±20%	20±20%	30±20%	40±20%	50±20%
vp	0.83	1.80	2.50	3.21	4.01
vpac	0.85	1.85	2.56	3.26	4.04

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

Algorithm	Pattern length				
	10±20%	20±20%	30±30%	40±40%	50±50%
vp	0.84	1.01	0.80	0.73	0.68
vpac	0.61	0.89	0.70	0.64	0.59

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

$$n_{vp}/N \text{ and } n_{vpac}/N. \quad (4)$$

To measure the effectiveness of vpac in cutting branches/nodes, we define the *cut ratio* as

$$(n_{vp}-n_{vpac})/n_{vp}. \quad (5)$$

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±20% is very small (as compared to that of 20±20%). This is because l_{min} is small in the case of 10±20% 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_{vpac} and cut-ratio

	Pattern length				
	10±20%	20±20%	30±30%	40±40%	50±50%
N	215985	999934	999977	999969	999961
n_{vp}	58973	50662	33499	26011	21421
n_{vpac}	40789	41144	27608	21598	17896
n_{vp}/N	27.30%	5.07%	3.35%	2.60%	2.14%
n_{vpac}/N	18.88%	4.11%	2.76%	2.16%	1.79%
$(n_{vp}-n_{vpac})/n_{vp}$	30.83%	18.79%	17.58%	16.96%	16.45%

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 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_{min} as data points in the edit distance-based metric space for $1 \le i \le r$, we design two schemes which construct the data points of T into vp-tree and vpac-tree, repsectively, and search data points of P_i in the trees, for solving EMSMP. The expected tree constructing time is $O(\lceil l_{min}/\omega \rceil \times l_{min} \times n \log_2 n)$ and the expected searching time is $O(\lceil l_{min}/\omega \rceil \times l_{min} \times n \times \log_2 n)$ for either vp-tree or vp-ac-tree.

The experimental results show that the proposed schemes using vp-tree and vpac-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 vpac-tree.

References

[1] Aho, A.V. and Corasick, M.J. (1975) Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18, 333-340.

[2] Boyer, R. S. and Moore, J. S. (1977) A fast string searching algorithm. *Commun. ACM*, 20, 762-772.

[3] Commentz-Walter, B. (1979) A string matching algorithm fast on the average. *Proc. 6th Colloquium*, Graz, Austria, July 16-20, pp. 118-132. Springer, Berlin.

[4] Hyyrö, H. (2003) A bit-vector algorithm for computing Levenshtein and Damerau edit distances. *Nord. J. Comput.*, 10, 29-39.

[5] Knuth, D.E., Morris, J.H. and Pratt, V.R. (1977) Fast pattern matching in strings. *SIAM J. Comput.*, 6, 323-350.

[6] Levenshtein, V. I. (1965) Binary codes capable of correcting deletions, insertions, and reversals. *Dokl. Akad. Nauk SSSR+*, 163, 845-848.

[7] Navarro, G. and Raffinot, M. (1998) A bit-Parallel approach to suffix automata: fast extended string matching. *Proc. CPM 98*, Piscataway, NJ, USA, July 20–22, pp. 14–33. Springer, Berlin.

[8] Navarro, G. and Raffinot, M. (2000) Fast and flexible string matching by combining bit-parallelism and suffix automata. *ACM J. Exper. Algorithmics*, 5.

[9] Wagner, R. A. and Michael, J. F. The string-to-string correction problem. *J. ACM*, 21, 168-173.

[10] Wu, S. and Manber, U. (1994) A fast algorithm for multi-pattern searching. Technical Report TR-94-17. Department of Computer Science, University of Arizona, USA.

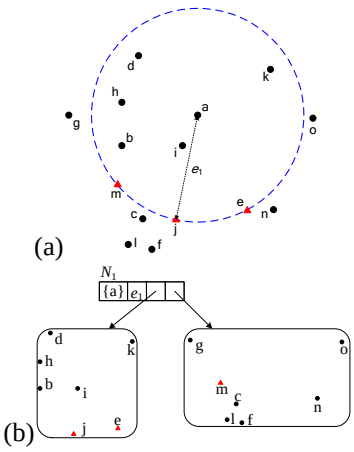
[11] Yianilos, P. N. (1993) Data structures and algorithms for nearest neighbor search in general metric spaces. *Proc. SODA 93*, Austin, Texas, USA, September, pp. 311-321. Society for Industrial and Applied Mathematics, Philadelphia.

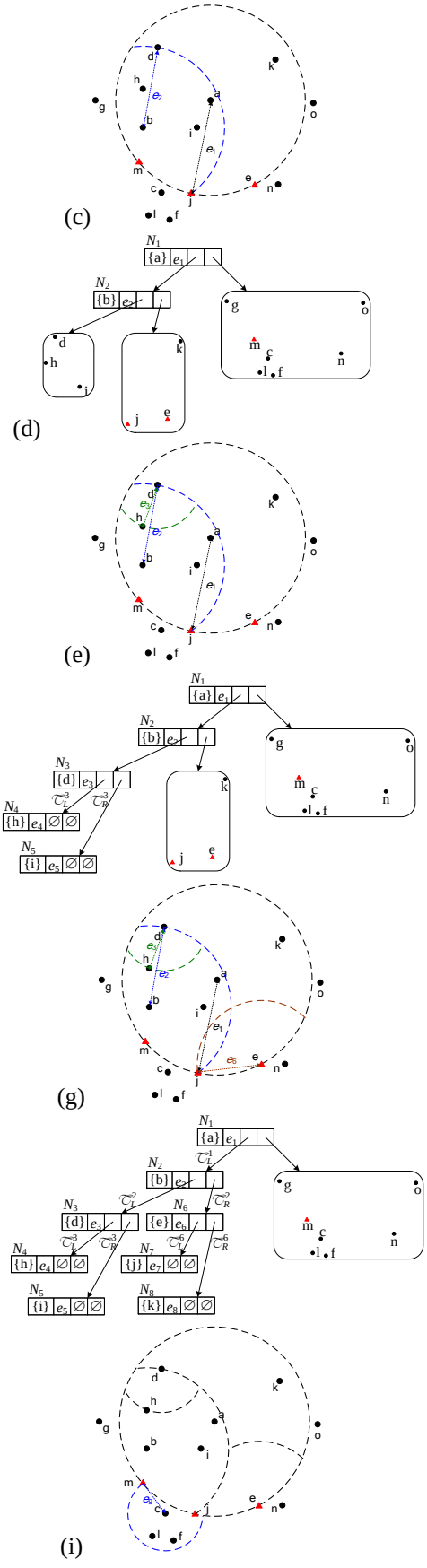
Appendix A

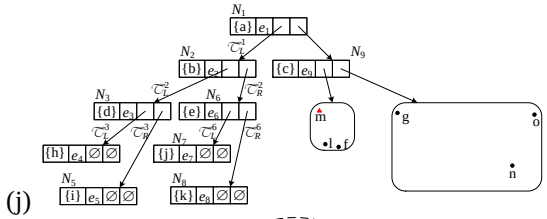
The construction of vp-tree with respect to X = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o}, as shown in Figure 2(a) is illustrated level by level as follows. In the

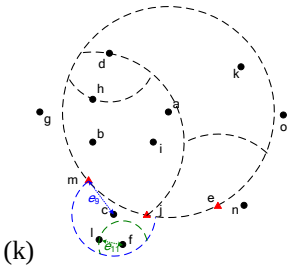
beginning, we choose a as the first vantage point and we have $X = X\backslash\{a\}$ and $V_1 = \{a\}$. Among all $\delta(a, x)$'s, the median distance is $e_1 = \delta(a, j)$ (= $\delta(a, e)$ = $\delta(a, m)$) as shown in Figure A(a) where e, j and m are denoted as red triangles. Based on median e_1, we apply the revised refined ball partition procedure to obtain X_L^1 = {i, k, b, d, h, e, j} and X_R^1 = {m, l, c, n, f, o, g}. The node with respect to X would be N^1 = ({a}, e_1, $\mathcal{T}_L^1$, $\mathcal{T}_R^1$) where $\mathcal{T}_L^1$ and $\mathcal{T}_R^1$ are the sub-trees with respect to X_L^1 and X_R^1, 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 (= X_L^1) = {i, k, b, d, h, e, j}. Assume that b is chosen to be its vantage point with $X^2 = X^2\backslash\{b\}$, $V_2 = \{b\}$ and median $e_2 = \delta(b, d) = \delta(b, j)$. Then, we obtain X_L^2 = {h, i, d} and X_R^2 = {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 $X^3 = X_L^2\backslash\{d\}$, $V_3 = \{d\}$ and median $e_3 = \delta(d, h)$. Hence, X_L^3 = {h} and X_R^3 = {i} as shown in Figure A(e). In node N^4 (N^5) with respect to $X^4 = X_L^3$ ($X^5 = X_R^3$), 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 $\mathcal{T}_L^3$ and $\mathcal{T}_R^3$ after the construction of N^4 and N^5, respectively, cause N^3 to be physically built. For N^6, vantage point is e and $e_6 = \delta(e, j)$ (Figure A(g)), which makes nodes N^7 and N^8 (containing j and k with pointers $\mathcal{T}_L^6$ and $\mathcal{T}_R^6$, respectively) be built as leaf nodes. $\mathcal{T}_L^6$ and $\mathcal{T}_R^6$ cause N^6 to be built with $\mathcal{T}_R^2$; and consequently, $\mathcal{T}_L^2$ and $\mathcal{T}_R^2$ (to N^3 and N^6, respectively) make N^2 be built with pointer $\mathcal{T}_L^1$ as shown in Figure A(h).

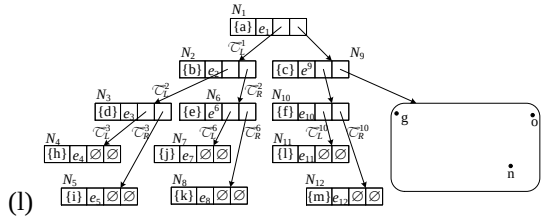
The sub-tree pointed by $\mathcal{T}_R^1$ is constructed by the same recursive approach. Figure A(i) and (j) depict that c is the vantage point of N^9 with $e_9 = \delta(c, m)$, X_L^9 = {f, l, m} and X_R^9 = {g, n, o}. Figure A (k) show vantage point f (l, m) in N^{10} (N^{11}, N^{12}) where N^{11} and 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^{14} and N^{15}; and A(n) presents the final vp-tree.

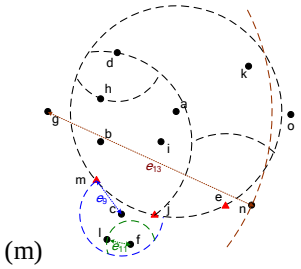












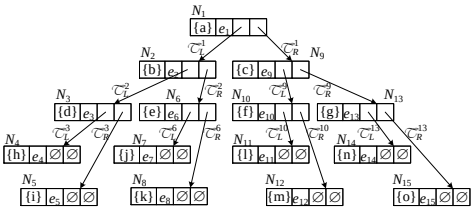


Figure A. The constructing vp-tree for Example 1