Improved Algorithms for Finding Edit Distance Based Motifs

Soumitra Pal[1] **and Sanguthevar Rajasekaran**[1,*]

[1]Computer Science and Engineering, University of Connecticut, 371 Fairfield Road, Storrs, CT 06269, USA.
[*]rajasek@engr.uconn.edu

ABSTRACT

Motif search is an important step in extracting meaningful patterns from biological data. Since the general problem of motif search is intractable, there is a pressing need to develop efficient exact and approximation algorithms to solve this problem. We design novel algorithms for solving the *Edit-distance-based Motif Search (EMS)* problem: given two integers l,d and n biological strings, find all strings of length l that appear in each input strings with at most d substitutions, insertions and deletions. These algorithms have been evaluated on several challenging instances. Our algorithm solves a moderately hard instance $(11,3)$ in a couple of minutes and the next difficult instance $(14,3)$ in a couple of hours whereas the best previously known algorithm, EMS1, solves $(11,3)$ in a few hours and does not solve $(13,4)$ even after 3 days. This significant improvement is due to a novel and provably efficient neighborhood generation technique introduced in this paper. This efficient approach can be used in other edit distance based applications in Bioinformatics, such as k-spectrum based sequence error correction algorithms. We also use a trie based data structure to efficiently store the candidate motifs in the neighbourhood and to output the motifs in a sorted order.

Introduction

Motif search has applications in solving such crucial problems as identification of alternative splicing sites, determination of open reading frames, identification of promoter elements of genes, identification of transcription factors and their binding sites etc. (see e.g., Nicolae and Rajasekaran[1]). There are many formulations of the motif search problem. A widely studied formulation is known as (l,d)-motif search or Planted Motif Search (PMS).[2] Given two integers l,d and n biological strings the problem is to find all short strings of length l that appear in each of the n input strings with at most d mismatches. There is a significant amount of work in the literature on PMS [1, 3, 4, and so on].

PMS considers only point mutations as events of divergence in biological sequences. However, insertions and deletions also play important roles in divergence.[2,5] Therefore, researchers have also considered a formulation in which the Levenshtein distance (or edit distance), instead of mismatches, is used for measuring the degree of divergence.[6,7] Given n strings $S^{(1)},S^{(2)},\ldots,S^{(n)}$, each of length m from a fixed alphabet Σ, and integers l,d, the Edit-distance-based Motif Search (EMS) problem is to find all patterns M of length l that occur in at least one position in each $S^{(i)}$ with an edit distance of at most d. More formally, M is a motif if and only if $\forall i$, there exist $k \in [l-d,l+d]$, $j \in [1,m-k+1]$ such that for the substring $S^{(i)}_{j,k}$ of length k at position j of $S^{(i)}$, $ED(S^{(i)}_{j,k},M) \leq d$. Here $ED(X,Y)$ stands for the edit distance between two strings X and Y.

EMS is also NP-hard since PMS is a special case of EMS and PMS is known to be NP-hard.[8] As a result, any exact algorithm for EMS that finds all the motifs for a given input can be expected to have an exponential (in some of the parameters) worst case runtime. One of the earliest EMS algorithms is due to Rocke and Tompa[6] and is based on Gibbs Sampling which requires repeated searching of the motifs in a constantly evolving collection of aligned strings, and each search pass requires $O(nl)$ time. Sagot[7] gave a suffix tree based algorithm that takes $O(n^2ml^d|\Sigma|^d)$ time and $O(n^2m/w)$ space where w is the word length of the computer. Adebiyi and Kaufmann[9] proposed an algorithm with an expected runtime of $O(nm+d(nm)^{(1+pow(\varepsilon))}\log nm)$ where $\varepsilon = d/l$ and $pow(\varepsilon)$ is an increasing concave function. The value of $pow(\varepsilon)$ is roughly 0.9 for protein and DNA sequences.

Rajasekaran et al.[10] proposed a simpler Deterministic Motif Search (DMS) that has the same worst case time complexity as the algorithm by Sagot.[7] The algorithm generates and stores the neighbourhood of every substring of length in the range $[l-d,l+d]$ of every input string and using a radix sort based method outputs the neighbours that are common to at least one substring of each input string. This algorithm was implemented by Pathak et al.[11]

Following a useful practice for PMS algorithms, Pathak et al.[11] evaluated their algorithm on certain instances that are considered challenging and are generated as follows: $n=20$ random DNA/protein strings of length $m=600$, and a short random string M of length l are generated according to the independent identically distributed (i.i.d) model. A separate random d-edit distance neighbour of M is "planted" in each of the n input strings. Such an (l,d) instance is defined to be a challenging instance if d is the largest integer for which the expected number of spurious motifs, i.e., the motifs of length l that would occur in the input by random chance, does not exceed a constant (10000 in this paper).

Table 4 shows the expected number of spurious motifs in the random instances for l in the range $[5,17]$ and for d in the range $[0,l-2]$ computed using equation (27) given in Appendix . Though the expected number of spurious motifs in EMS instances are in general different from those in PMS instances, it turns out that the challenging instances are the same for both problems for the range of l,d we considered: $(7,1),(9,2),(11,3),(13,4)$, and $(15,5)$.

The algorithm by Pathak et al.[11] solves the moderately hard instance $(11,3)$ in a few hours and does not solve the next difficult instance $(13,4)$ even after 3 days. A key time consuming part of the algorithm is in the generation of the edit distance neighbourhood of all subsequences as there are many common neighbours.

Contributions

In this paper we present an improved algorithm for EMS that solves the instance $(11,3)$ in less than a couple of minutes and the instance $(14,3)$ in less than a couple of hours. Our algorithm uses an efficient technique (introduced in this paper) to generate the edit distance neighbourhood of length l with distance at most d of all substrings of an input string.

We prove that every substring identified by our neighborhood generation technique as a neighbor of a string is nearly distinct. In other words, our neighborhood generation technique does not spend a lot of time generating neighbors that have already been generated. This efficient approach can be used in other edit distance based applications in Bioinformatics, such as generating the candidate set of corrections in k-spectrum based error correction algorithms[12] for sequences with insertion and deletion errors.

We use a trie based data structure to efficiently store the neighbourhood. This not only simplifies the removal of duplicate neighbours but also helps in outputting the final motifs in sorted order using a depth first search traversal of the trie.

A C++ implementation of our algorithm is available at `https://github.com/soumitrakp/ems2.git`.

Methods

In this section we introduce some notations and observations.

An (l,d)-*friend* of a k-mer L is an l-mer that is at an exact distance of d from L. Let $F_{l,d}(L)$ denote the set of all (l,d)-friends of L. An (l,d)-*neighbour* of a k-mer L is an l-mer that is at a distance of no more than d from L. Let $N_{l,d}(L)$ denote the set of all (l,d)-neighbours of L. Then

$$N_{l,d}(L) = \cup_{d'=0}^{d} F_{l,d'}(L). \tag{1}$$

For a string S of length m, an (l,d)-*motif* of S is an l-mer that is at a distance no more than d from some substring of S. Thus an (l,d)-motif of S is an (l,d)-neighbour of at least one substring $S_{j,k} = S_j S_{j+1} \dots S_{j+k-1}$ where $k \in [l-d, l+d]$. Therefore, the set of (l,d)-motifs of S, denoted by $M_{l,d}(S)$, is given by

$$M_{l,d}(S) = \cup_{k=l-d}^{l+d} \cup_{j=1}^{m-k+1} N_{l,d}(S_{j,k}). \tag{2}$$

For a collection of strings $\mathscr{S} = \{S^{(1)}, S^{(2)}, \dots, S^{(m)}\}$, a (common) (l,d)-motif is an l-mer that is at a distance no more than d from at least one substring of each $S^{(i)}$. Thus the set of (common) (l,d)-motifs of $\mathscr{S}$, denoted by $M_{l,d}(\mathscr{S})$, is given by

$$M_{l,d}(\mathscr{S}) = \cap_{i=1}^{n} M_{l,d}(S^{(i)}). \tag{3}$$

One simple way of computing $F_{l,d}(L)$ is to grow the friendhood of L by one distance at a time for d times and to select only the friends having length l. Let $G(L)$ denote the set of strings obtained by one edit operation on L and $G(\{L_1, L_2, \dots, L_r\}) = \cup_{t=1}^{r} G(L_t)$. If $G^1(L) = G(L)$, and for $t > 1$, $G^t(L) = G(G^{t-1}(L))$ then

$$F_{l,d}(L) = \{x \in G^d(L) : |x| = l\}. \tag{4}$$

Using equations (1), (2), (3) and (4), Pathak et al.[11] gave an algorithm that stores all possible candidate motifs in an array of size $|\Sigma|^l$. However the algorithm is inefficient in generating the neighbourhood as the same candidate motif is generated by several combinations of the basic edit operations. Also, the $O(|\Sigma|^l)$ memory requirement makes the algorithm inapplicable for larger instances. In this paper we mitigate these two limitations.

Efficient Neighbourhood Generation

We now give a more efficient algorithm to generate the (l,d)-neighbourhood of all possible k-mers of a string. Instead of computing (l,d')-friendhood for all $0 \le d' \le d$, we compute only the exact (l,d)-friendhood.

Lemma 1. $M_{l,d}(S) = \cup_{k=l-d}^{l+d} \cup_{j=1}^{m-k+1} F_{l,d}(S_{j,k})$.

Proof. Consider the k-mer $L = S_{j,k}$. If $k = l-d$ then we need d insertions to make L an l-mer. There cannot be any (l,d')-neighbour of L for $d' < d$. Thus

$$\cup_{d'=0}^{d} F_{l,d'}(S_{j,l-d}) = F_{l,d}(S_{j,l-d}). \tag{5}$$

Suppose $k > l-d$. Any $(l,d-1)$-neighbour B of L is also an (l,d)-neighbour of $L' = S_{j,k-1}$ because $ED(B,L') \le ED(B,L) + ED(L,L') \le (d-1)+1 = d$. Thus

$$\cup_{d'=0}^{d} F_{l,d'}(S_{j,k}) \subseteq F_{l,d}(S_{j,k}) \bigcup \cup_{d'=0}^{d} F_{l,d'}(S_{j,k-1})$$

which implies that

$$\cup_{k'=k}^{k-1} \cup_{d'=0}^{d} F_{l,d'}(S_{j,k'}) = F_{l,d}(S_{j,k}) \bigcup \cup_{d'=0}^{d} F_{l,d'}(S_{j,k-1}). \tag{6}$$

Applying equation (6) repeatedly for $k = l+d, l+d-1, \ldots, l-d+1$, along with equation (5) in equations (1) and (2) gives the result of the lemma. □

We generate $F_{l,d}(S_{j,k})$ in three phases: we apply δ deletions in the first phase, β substitutions in the second phase, and finally α insertions in the final phase, where $d = \delta + \alpha + \beta$ and $l = k - \delta + \alpha$. Solving for α, β, δ gives $\max\{0,q\} \le \delta \le (d+q)/2$, $\alpha = \delta - q$ and $\beta = d - \alpha - \delta = d - 2\delta + q$ where $q = k-l, q \in [-d,+d]$. In each of the phases, the neighbourhood is grown by one edit operation at a time. Duplication in the friendhood is avoided as follows: if the current edit operation is at position j, the subsequent edit operation in the same phase are restricted at positions larger than j. Formally,

$$F_{l,d}(L) = \cup_{\delta \in [\max\{0,q\},(d+q)/2]} G(L,1,\delta,d-2\delta+q,\delta-q) \tag{7}$$

where $q = |L| - l$ and $G(L,j,\delta,\beta,\alpha)$

$$= \begin{cases} \cup_{j'=j}^{|L|} G(del(L,j'),j',\delta-1,\beta,\alpha) & \text{if } \delta > 0 \\ H(L,1,\beta,\alpha) & \text{otherwise} \end{cases} \tag{8}$$

where $del(L,j)$ is the string obtained by deleting L_j and $H(L,j,\beta,\alpha) =$

$$\begin{cases} \cup_{j'=j}^{|L|} \cup_{\sigma \ne L_j} H(sub(L,j',\sigma),j'+1,\beta-1,\alpha) & \text{if } \beta > 0 \\ I(L,1,\alpha) & \text{otherwise} \end{cases} \tag{9}$$

where $sub(L,j,\sigma)$ is obtained by substituting L_j by σ and $I(L,j,\alpha)$

$$= \begin{cases} \cup_{j'=j}^{|L|} \cup_{\sigma} I(ins(L,j',\sigma),j'+1,\alpha-1) & \text{if } \alpha > 0 \\ L & \text{otherwise} \end{cases} \tag{10}$$

where $ins(L,j,\sigma)$ is obtained by inserting σ just before L_j.

Compact Motifs

Let us consider an example 3-mer $\sigma_1\sigma_2\sigma_3$ and all possible single insertions just before position 2. The $|\Sigma|$ neighbours thus generated can be compactly represented as $\sigma_1 * \sigma_2\sigma_3$ where $*$ represents any character in Σ. Similarly a substitution at position 2 can be compactly represented by $\sigma_1 * \sigma_3$ where $*$ represents any character in $\Sigma \setminus \sigma_2$. This can be generalized for any number of substitutions and insertions.

Let Σ' represent the compact alphabet $\Sigma \cup \{*\}$ where $*$ is a special character not included in Σ. The *expansion* of a compact string $L \in \Sigma'^{+}$ is defined as $X(L) = \{T \in \Sigma^{|L|} \mid T_j \ne L_j \Rightarrow L_j = *\}$. The representation of a set of strings in Σ^{+} using compact strings in Σ'^{+} is not unique. For example, if $\Sigma = \{0,1\}$, the set $\{000,001,100,101\}$ can be represented by $\{00*,100,101\}$,

$\{00*, 10*\}$, $\{*0*\}$, and so on. Here we are interested in only those compact strings that arise if we use the special character $*$ for insertion and substitution. Let the compact friendhood thus generated be denoted as $\bar{F}_{l,d}(L)$ and formally defined as

$$\bar{F}_{l,d}(L) = \cup_{\delta \in [\max\{0,q\},(d+q)/2]} \bar{G}(L, 1, \delta, d - 2\delta + q, \delta - q) \tag{11}$$

where $q = |L| - l$ and $\bar{G}(L, j, \delta, \beta, \alpha)$

$$= \begin{cases} \cup_{j'=j}^{|L|} \bar{G}(del(L, j'), j', \delta - 1, \beta, \alpha) & \text{if } \delta > 0 \\ \bar{H}(L, 1, \beta, \alpha) & \text{otherwise} \end{cases} \tag{12}$$

$$\bar{H}(L, j, \beta, \alpha) = \begin{cases} \cup_{j'=j}^{|L|} \bar{H}(sub(L, j', *), j'+1, \beta-1, \alpha) & \text{if } \beta>0 \\ \bar{I}(L, 1, \alpha) & \text{otherwise} \end{cases} \tag{13}$$

$$\bar{I}(L, j, \alpha) = \begin{cases} \cup_{j'=j}^{|L|} \bar{I}(ins(L, j', *), j'+1, \alpha-1) & \text{if } \alpha > 0 \\ L & \text{otherwise.} \end{cases} \tag{14}$$

For $\mathscr{S} = \{S^{(1)}, S^{(2)}, \ldots, S^{(m)}\}$, let

$$\bar{M}_{l,d}(\mathscr{S}) = \cap_{i=1}^{n} \bar{M}_{l,d}(S^{(i)}) \quad \text{where} \tag{15}$$

$$\bar{M}_{l,d}(S) = \cup_{k=l-d}^{l+d} \cup_{j=1}^{m-k+1} \bar{F}_{l,d}(S_{j,k}) \tag{16}$$

with the usual definition of union and the following definition of intersection on compact strings $A, B \in \Sigma'^l$

$$A \cap B = \begin{cases} \emptyset & \text{if } \exists j \text{ s.t. } A_j, B_j \in \Sigma, A_j \neq B_j \\ \sigma_1\sigma_2\ldots\sigma_l \text{ otherwise where } \sigma_j = \begin{cases} b_j & \text{if } a_j = * \\ a_j & \text{if } b_j = * \end{cases} \end{cases} \tag{17}$$

and on sets of compact strings in Σ'^l

$$\{A^{(1)}, A^{(2)}, \ldots\} \cap \{B^{(1)}, B^{(2)}, \ldots\} = \cup_{i,j}(A^{(i)} \cap B^{(j)}). \tag{18}$$

Lemma 2. $M_{l,d}(\mathscr{S}) = X(\bar{M}_{l,d}(\mathscr{S}))$.

Proof. The equivalence is straight forward if there are only insertions. For substitution too the equivalence is valid as L is already in the neighbourhood and $L \cup_{\sigma' \neq \sigma} sub(L, j', \sigma') \equiv sub(L, j', *)$. □

Trie for Storing Compact Motifs

We store the compact motifs $\bar{M}_{l,d}(S)$ in a trie based data structure which we call a *motif trie*. This helps implement the intersection defined in equation (17) and required in equation (15). Each node in the motif trie has at most $|\Sigma|$ children. The edges from a node u to its children v are labeled with mutually exclusive subsets $label(u, v) \subseteq \Sigma$. An empty set of compact motifs is represented by a single root node. A non-empty trie has $l + 1$ levels of nodes, the root being at level 0. The trie stores the l-mer $\sigma_1\sigma_2\ldots\sigma_l$, all $\sigma_j \in \Sigma$, if there is a path from the root to a leaf where σ_j appears in the label of the edge from level $j - 1$ to level j.

For each string $S = \mathscr{S}^{(i)}$ we keep a separate motif trie $M^{(i)}$. Each compact neighbour $A \in \bar{F}_{l,d}(S_{j,k})$ generated by equation (11) is inserted into the motif trie recursively as follows. We start with the root node where we insert $A_1A_2\ldots A_l$. At a node u at level j where the prefix $A_1A_2\ldots A_{j-1}$ is already inserted, we insert the suffix $A_jA_{j+1}\ldots A_l$ as follows. If $A_j \in \Sigma$ we insert $A' = A_{j+1}A_{j+2}\ldots A_l$ to the children v of u such that $A_j \in label(u, v)$. If $label(u, v) \neq \{A_j\}$, before inserting we make a copy of sub trie rooted at v. Let v' be the root of the new copy. We make v' a new child of u, set $label(u, v') = \{A_j\}$, remove A_j from $label(u, v)$, and insert A' to v'. On the other hand if $A_j = *$ we insert A' to each children of u. Let $T = \Sigma$ if $A_j = *$ and $T = \{A_j\}$ otherwise. Let $R = T \setminus \cup_v label(u, v)$. If $T \neq \emptyset$ we create a new child v' of u, set $label(u, v') = R$ and recursively insert A' to v'. Fig. 1 shows examples of inserting into the motif trie.

We also maintain a motif trie for the $\mathscr{M}$ for the common compact motifs found so far, starting with $\mathscr{M} = M^{(1)}$. After processing string $S^{(i)}$ we intersect the root of $M^{(i)}$ with the root of $\mathscr{M}$. In general a node $u_2 \in M^{(i)}$ at level j is intersected with a node $u_1 \in \mathscr{M}$ at level j using the procedure shown in Algorithm 1. Fig. 2 shows an example of the intersection of two motif tries.

The final set of motifs $X(\bar{M}_{l,d}(\mathscr{S}))$ is obtained by a depth-first traversal of $\mathscr{M}$ outputting the label of the path from the root whenever a leaf is traversed. An edge (u, v) is traversed separately for each $\sigma \in label(u, v)$.

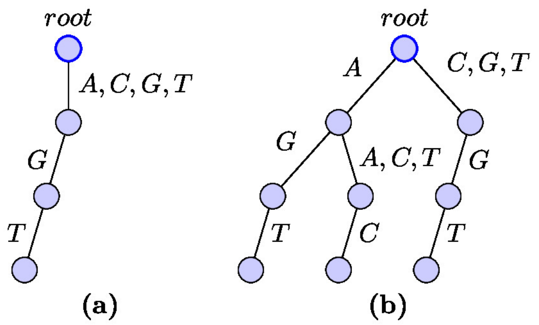

Figure 1. Inserting into motif trie for $\Sigma = \{A, C, G, T\}$ and $l = 2$. (a) After inserting $*GT$ into empty trie. (b) After inserting another string $A*C$.

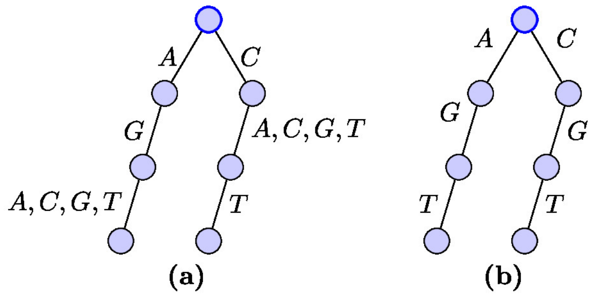

Figure 2. Intersection of motif tries. (a) Trie for $AG* \cup C*T$. (b) Intersection of trie in Fig. 1(b) and trie in Fig. 2(a).

Algorithm 1: Intersect sub tries u_1 and u_2 with result in u_1

```
$V \leftarrow$ all children of $u_1$;
foreach $v_1 \in V$ do
    foreach child $v_2$ of $u_2$ do
        $newLabel \leftarrow label(u_1, v_1) \cap label(u_2, v_2)$;
        if $newLabel \neq \emptyset$ then
            if $newLabel \neq label(u_1, v_1)$ then
                let $v'_1$ be a new child of $u_1$;
                copy at $v'_1$ the sub trie rooted at $v_1$;
                $label(u, v'_1) \leftarrow newLabel$;
                $label(u, v_1) \leftarrow label(u, v_1) \setminus newLabel$;
                recursively intersect $v'_1$ with $v_2$;
            else
                recursively intersect $v_1$ with $v_2$;
    if $label(u_1, v_1) = \emptyset$ then delete sub trie rooted at $v_1$;
if $u_1$ has no child then delete sub trie rooted at $u_1$;
```

Efficient Compact Neighbourhood Generation

The set of compact motifs $\bar{M}_{l,d}(S)$ computed in equation (15) is in fact a multiset as the compact motifs generated are not all distinct. A significant part of the time taken by our algorithm is in inserting compact neighbours into the motif trie as it is executed for each neighbour in the friendhood. We improve the performance of our algorithm using some simple rules to reduce the number of times a compact motif is generated. Later we will see that these rules are quite close to the ideal as we will prove that the compact motif generated after skipping using the rules, are distinct if all the characters in the string are distinct.

To differentiate multiple instances of the same element in the multiset $\bar{M}_{l,d}(S)$ we augment it with the information about how it is generated by equation (15). Formally, each instance L of an element in $\bar{M}_{l,d}(S)$ is represented as an ordered tuple $\langle M, S_{j,k}, T\rangle$ where the sequence of edit operations T when applied to $S_{j,k}$ gives M. Each edit operation in T is represented as a tuple $\langle p,t\rangle$ where p denotes the position in S where the edit operation is applied and $t \in \{D,R,I\}$ denotes the type of the operation - deletion, substitution and insertion, respectively. At each position there can be one deletion or one substitution but one or more insertions. The tuples in T are sorted lexicographically with the normal order for p and for t, $D < R < I$. In the rest of the paper we assume that $\bar{M}_{l,d}(S)$ is a set of tuples $\langle M, S_{j,k}, T\rangle$.

The rules for skipping the elements of the multiset $\bar{M}_{l,d}(S)$ are shown in Table 1. We first show that the rules do not exclude any necessary compact motif. Let $\bar{\bar{M}}_{l,d}(S)$ be the set of compact motifs generated by equation (15) except those excluded by Rules 1-9. Let $\Gamma(\langle M, S_{j,k}, T\rangle) = M$ and $\Gamma(Z) = \cup_{L\in Z}\Gamma(L)$.

Lemma 3. $\Gamma(\bar{\bar{M}}_{l,d}(S)) = \Gamma(\bar{M}_{l,d}(S))$.

Proof. By construction, $\Gamma(\bar{\bar{M}}_{l,d}(S)) \subseteq \Gamma(\bar{M}_{l,d}(S))$. We show by contradiction that $\Gamma(\bar{M}_{l,d}(S)) \subseteq \Gamma(\bar{\bar{M}}_{l,d}(S))$.

Let $L_1 = \langle M_1, S_{j_1,k_1}, T_1\rangle$ and $L_2 = \langle M_2, S_{j_2,k_2}, T_2\rangle$ be two elements of $\bar{M}_{l,d}(S)$ and $\langle p_1,t_1\rangle \in T_1, \langle p_2,t_2\rangle \in T_2$ be the leftmost edit operations where T_1, T_2 differ. We impose an order $L_1 < L_2$ if and only if $(k_1 < k_2) \vee ((k_1 = k_2) \wedge (p_1 < p_2)) \vee ((k_1 = k_2) \wedge (p_1 = p_2) \wedge (t_1 < t_2))$.

Let $L = \langle M, S_{j,k}, T\rangle$ be the largest (in the order defined above) element in $\bar{M}_{l,d}(S)$ such that there is no element $L'' \in \bar{\bar{M}}_{l,d}(S)$ and $\Gamma(L'') = M$. For each Rule 1-9 we show a contradiction that if L is skipped by the rule then there is another $L' = \langle M, S_{j',k'}, T'\rangle \in \bar{M}_{l,d}(S)$ with the same number of edit operations but $L < L'$. Fig. 3 illustrates the choice of L' under different rules.

Rule 1. Here $j+k \le m$ and $\langle j,D\rangle \in T$. Consider $T' = T \setminus \langle j,D\rangle) \cup \langle j+k,D\rangle$, and $j' = j+1, k' = k$.

Rule 2. Consider $T' = T \setminus \{\langle j+t,D\rangle, \langle j+t+1,R\rangle\} \cup \{\langle j+t,R\rangle, \langle j+t+1,D\rangle\}$, and $j' = j, k' = k$.

Rule 3. $T' = T \setminus \{\langle j,R\rangle, \langle j{+}t{+}1,D\rangle\} \cup \{\langle j{+}t{+}1,R\rangle, \langle j{+}k,D\rangle\}$, $j' = j+1, k' = k$.

Rule 4. For this and subsequent rules $k < l+d$ as there is at least one insertion and hence k' could possibly be equal to $k+1$. We consider two cases. Case (i) $j+k \le m$: $T' = T \setminus \{\langle j+t,D\rangle, \langle j+t,I\rangle\} \cup \{\langle j{+}t,R\rangle, \langle j+k,D\rangle\}$, $j' = j, k' = k+1$. Case (ii) $j+k = m+1$: Here deletion of S_j is allowed by Rule 1. $T' = T \setminus \{\langle j+t,D\rangle, \langle j+t,I\rangle\} \cup \{\langle j-1,D\rangle, \langle j{+}t,R\rangle\}$, $j' = j-1, k' = k+1$.

Rule 5. The same argument for Rule 4 applies considering $\langle j{+}t{+}1,I\rangle$ instead of $\langle j{+}t,I\rangle$.

Rule 6. $T' = T \setminus \langle j{+}t,I\rangle \cup \langle j{+}t{+}1,I\rangle$, and $j' = j, k' = k$.

Rule 7. $T' = T \setminus \langle j,I\rangle \cup \langle j{-}1,R\rangle$, $j' = j-1, k' = k+1$.

Rule 8. $T' = T \setminus \langle j{+}t,I\rangle \cup \langle j{-}1,R\rangle$, $j' = j-1, k' = k+1$.

Rule 9. $T' = T \setminus \langle j{+}k,I\rangle \cup \langle j{+}k,R\rangle$, $j' = j, k' = k+1$. □

Lemma 4. *If S_js are all distinct then for any two distinct $L, L' \in \bar{\bar{M}}_{l,d}(S)$, we have $\Gamma(L) \neq \Gamma(L')$.*

Proof. We prove by contradiction. Let $L = \langle M, S_{j,k}, T\rangle$ and $L' = \langle M, S_{j',k'}, T'\rangle$ be two distinct elements of $\bar{M}_{l,d}(S)$. We start with the four strings $O = S_{j,k}, O' = S_{j',k'}, N = N' = M$ and align them as follows. The characters common to all four are first aligned. For each S_p present only in one of O, O', we insert S_p in the other and insert $-$ in one of N, N', as appropriate. For each $\langle p,t\rangle \in T$, if $t = I$ we insert a $-$ at the corresponding position in O, O', N'. If $t = D$ we insert a $-$ at the corresponding position in N. If $t = R$ we align the corresponding $*$ in N with S_p in O. We repeat the analogous for $\langle p,t\rangle \in T'$ but making sure that only a single $-$ is inserted if both T, T' have an insertion at the same position, or both T, T' have a deletion at the same position.

Without loss of generality, assume $j \le j'$. If $j < j'$ then all of $S_j, S_{j+1}, \ldots, S_{j'-1}$ are either deleted or substituted in N. If $S_{j,k}$ is the not the rightmost k-mer of S then by Rule 1, S_j cannot be deleted in N and hence must be substituted. Then by Rule 3 all of $S_{j+1}, \ldots, S_{j'-1}$ are also substituted in N. Since the leftmost non $-$ character in N' must be $*$, $S_{j',k'}$ must be substituted in N' because by Rule 7 no insertion is possible just before $S_{j'}$ in N'. Since $S_{j'}$ cannot be deleted in N by Rule 3, $S_{j'}$ must be substituted in N. This implies there must be another $*$ just after the alignment of $S_{j'}$ in N'. Since by Rule 8, this $*$ cannot be due to an insertion, $S_{j'+1}$ must be substituted in N' which enforces $S_{j'+1}$ to be substituted in N. By repeating this argument, all characters in N would be $*$ which is not possible. Thus either $j = j'$ or $S_{j,k}$ is the rightmost k-mer of S. If $j \neq j'$ then $S_{j,k}$ must be rightmost and if S_j is substituted in N then a similar argument will show a contradiction. Thus in such a case S_j is deleted in

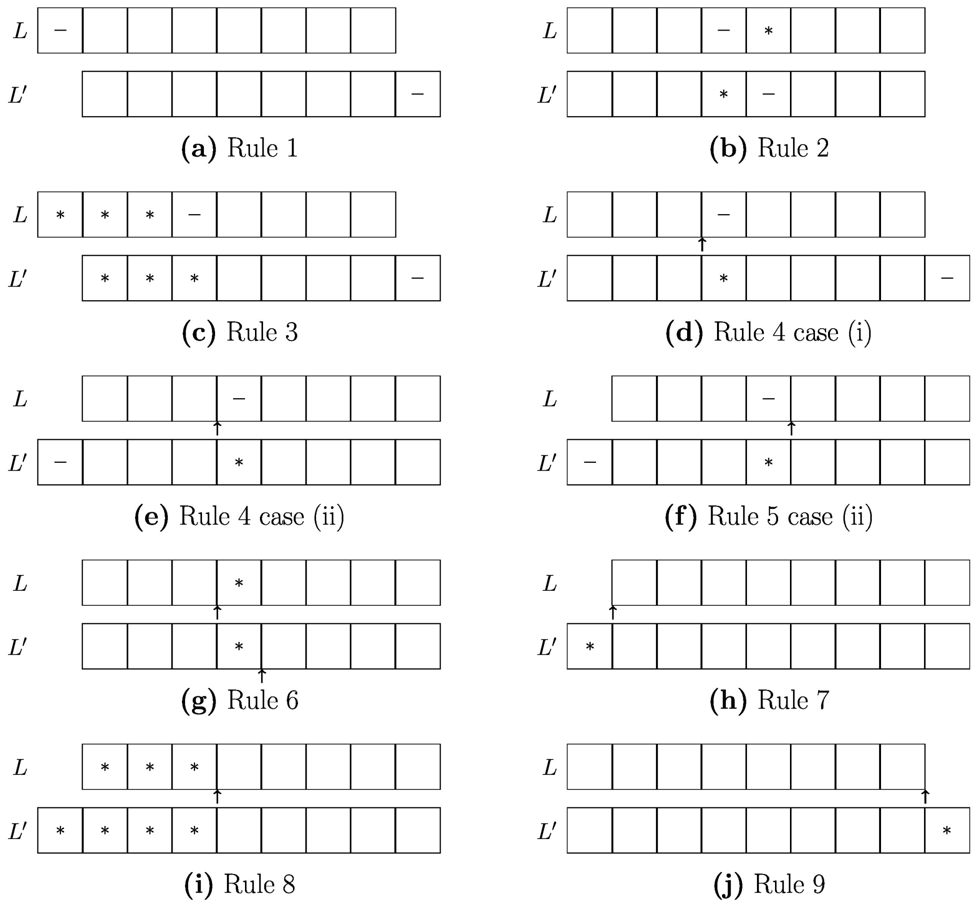

Figure 3. Construction of L' under different rules in the proof of Lemma 3. Insertions are shown using arrows, deletions using $-$ and substitutions using $*$. Rule 5 case (i) is similar to Rule 4 case (i) and omitted to save space.

N and by Rule 2, each of $S_{j+1},\ldots,S_{j'-1}$ is deleted in N. By the construction of the alignment there is a $-$ at the corresponding positions in N'. Thus in both cases, $j = j'$ and $j < j'$, we have the pattern $O_p = O'_p = S_p$ and $N_p = N'_p = -$ in the alignment for all $p < j'$.

Without loss of generality, we assume $j+k \leq j'+k'$ as we can always interchange $j+k$ and $j'+k'$ in the following argument. If $j+k < j'+k'$, let p be the leftmost in $[j+k+1, j'+k'-1]$ such that S_p is deleted in N'. Then by Rule 2, all of $S_{p+1}, S_{p+2}, \ldots, S_{j'+k'-1}$ must be deleted in N'. If $p > j+k$ or no such p exists then there is at least one $*$ on the right of the alignment of S_{j+k-1} in N'. By Rule 9, S_{j+k-1} must be substituted in N and hence by Rule 2, S_{j+k-1} must be substituted in N', and so on. This will imply all characters is N are $*$ which is not possible. Thus $p \leq j+k$ and in both cases $j+k = j'+k'$ and $j+k < j'+k'$, we have the pattern $O_p = O'_p = S_p$ and $N_p = N'_p = -$ in the alignment for all $p \geq j+k$.

Now let p be the leftmost position in the alignment where at least one of the two pairs O_p, O'_p and N_p, N'_p differs. In general, O_p, O'_p are either both equal to $\sigma \in \Sigma$ or both equal to $-$ and $N_p, N'_p \in \Sigma \cup \{*, -\}$. Table 2 shows the 8 possible restricted values for O_p, N_p, O'_p, N'_p under the assumption of p. Table 2 further shows why the cases $1,2,3,5,6,8$ are not possible. In case 4, to match $N_p = *$ there must be some $p' > p$ such that $N'_{p'} = *$. By Rules 2 and 5, $p' \neq p+1$. Thus $p' > p+1$ and $N'_{p+1}, N'_{p+2}, \ldots, N'_{p'-1}$ all must be $-$. By Rule 2, these $-$s can not be due to deletions in $S_{j',k'}$ and hence must be due to insertions in $S_{j,k}$. Since insertions to both $S_{j,k}, S_{j',k'}$ at the same position are aligned together, $N'_{p'} = *$ can not be due to an insertion in $S_{j',k'}$, it must be due a substitution in $S_{j',k'}$. The corresponding character in $S_{j,k}$ must be either deleted or substituted. Both are not possible due to Rules 4 and 6 respectively.

In case 7 too, to match $N_p = *$ there must be some $p' > p$ such that $N'_{p'} = *$. If $p' = p+1$ then $N'_{p'} = *$ must be due to some substitution in $S_{j',k'}$ and the corresponding character in $S_{j,k}$ must be either deleted or substituted which are not possible by Rules 4 and 6 respectively. Hence $p' > p+1$ and $N'_{p+1}, N'_{p+2}, \ldots, N'_{p'-1}$ all must be $-$. These $-$s can not be due to deletions in $S_{j',k'}$ because in that case the corresponding characters in $S_{j,k}$ must be either deleted or substituted which are not possible due to Rules 4 and 6 respectively. Thus $N'_{p+1}, N'_{p+2}, \ldots, N'_{p'-1}$ must be due to insertions in $S_{j,k}$. A similar argument as used in case 4 shows a contradiction in this case too. □

Algorithm 2: *genAll*(S)

```
foreach $q \leftarrow -d$ to $+d$ do
    $k \leftarrow l+q$;    $start \leftarrow 2$ ;                                    // Rule 1
    $leftMost \leftarrow rightMost \leftarrow$ false;
    for $j \leftarrow 1$ to $|S|-k+1$ do
        if $j=1$ then $leftMost \leftarrow$ true;
        if $j+k-1=m$ then $rightMost \leftarrow$ true; $start \leftarrow 1$;
        foreach $\delta \leftarrow \max\{0,q\}$ to $(d+q)/2$ do
            $\bar{\bar{G}}(S_{j,k}, start, \delta, d-2\delta+q, \delta-q)$;
```

Algorithm 3: $\bar{\bar{G}}(L, j, \delta, \beta, \alpha)$

```
if $\delta = 0$ then $\bar{\bar{H}}(L, j, \beta, \alpha)$; return;
foreach $j' \leftarrow j$ to $|L|$ do
    $\bar{\bar{G}}(sub(L, j', -), j'+1, \delta-1, \beta, \alpha)$;
```

In general the S_js are not distinct. However, as the input strings are random, the repetitions due to repeated characters are limited. Our experimentation shows that on an average each compact motif in $\bar{\bar{M}}_{l,d}(S)$ is repeated c times where c is a real number in $[1,2]$. Thus we can safely say that Rules 1-9 make $\bar{\bar{M}}_{l,d}(S)$ almost duplicate free.

Implementation: To track the deleted characters, instead of actually deleting we substitute them by a new symbol $-$ not in Σ'. We populate the motif trie $M^{(i)}$ by calling $genAll(S^{(i)})$ given in Algorithm 2. Rules 1-8 are incorporated in $\bar{\bar{G}}(L, j, \delta, \beta, \alpha)$, $\bar{\bar{H}}(L, j, \beta, \alpha)$ and $\bar{\bar{I}}(L, j, \alpha)$ which are shown in Algorithms 3, 4, and 5, respectively.

Results

We have implemented our sequential algorithm in C++ and evaluated on a Dell Optiplex 7020 desktop with Intel i5-4590 CPU at 3.30GHz and 8GB RAM running Linux Mint 17 (Ubuntu 14.04) OS. We generated random (l,d) instances according to Pevzner and Sze[2] and as described in the introduction. For every (l,d) combination we report the average runtime over 5 random instances. We compare our algorithm EMS2 with a modified implementation of the algorithm EMS1[11] which considered the neighbourhood of only l-mers whereas the modified version considers the neighbourhood of all k-mers where $l-d \leq k \leq l+d$.

A comparison between the runtime and the memory usage of the two algorithms are given in Table 3. Our efficient procedure to generate neighbourhood enables our algorithm to solve the instance $(13,4)$ in less than two hours which EMS1 could not solve even in 3 days. For the instance $(13,4)$, the memory used by EMS1 is computed using extrapolation $(91*(91/3))$ as EMS1 did not complete in the stipulated time. Note that the factor by which EMS2 takes more memory compared to EMS1 gradually decreases as the instances become harder. Our current implementation of EMS2 stores 4 child pointers in each node of the motif trie corresponding to each edge label A,C,G,T. We are working on an implementation in which we need two pointers: one for the leftmost child and one for the immediate right sibling. This is expected to reduce memory usage by about a half for DNA sequences and significantly for protein sequences.

Conclusions

We have presented an efficient algorithm for the EMS problem. Our algorithm is able to efficiently generate neighbourhood using some novel and elegant rules to avoid multiple instances of the same motif being generated in the neighbourhood. We have also proved that these rules are close to the ideal in the sense that the neighbourhood generated by our algorithm is distinct if the characters in the string are distinct. Though this condition is not practical and ideas by Knuth[13] can be used to generate distinct neighbourhood even when the characters in the string are repeated, nevertheless the rules help because the instances are randomly generated and hence the number of times a k-mer appears in any input string is very small. The second reason for the efficiency of our algorithm is the use of a trie based data structure to efficiently and compactly store the motifs.

References

1. Nicolae, M. & Rajasekaran, S. qPMS9: An Efficient Algorithm for Quorum Planted Motif Search. *Nature Scientific Reports* **5** (2015).

Algorithm 4: $\bar{\bar{H}}(L, j, \beta, \alpha)$

```
if $\beta = 0$ then
    $t \leftarrow \begin{cases} \text{largest } t' & \text{s.t. } L_{j'} = * \text{ for all } j' \le t' \\ 0 & \text{if no such } j' \text{ exists} \end{cases}$;
    $start \leftarrow \begin{cases} 1 & \text{if } leftMost \\ t+2 & \text{otherwise} \end{cases}$;                // Rules 7,8
    $\bar{\bar{I}}(L, start, \alpha)$; return;
foreach $j' \leftarrow j$ to $|L|$ do
    if $L_j = -$ then continue;                                       // already deleted
    if $(j > 1) \wedge L_{j-1} = -$ then continue;                    // Rule 2
    if $\neg rightMost \wedge_{j''<j'} (L_{j''} = *) \wedge (L_{j'+1} = -)$ then
        continue ;                                                    // Rule 3
    $\bar{\bar{H}}(sub(L, j', *), j'+1, \beta-1, \alpha)$;
```

Algorithm 5: $\bar{\bar{I}}(L, j, \alpha)$

```
if $\alpha = 0$ then insert $L$ to $M^{(i)}$ after deleting all $-$ in $L$; return;
foreach $j' \leftarrow j$ to $|L|$ do
    if $L_j \in \{-, *\}$ then continue;                              // Rules 4,6
    if $(j > 1) \wedge (L_{j-1} = -)$ then continue;                  // Rule 5
    $\bar{\bar{I}}(ins(L, j', *), j'+1, \alpha-1)$;
if $rightMost \wedge (L_{|L|} \neq -)$ then
    $\bar{\bar{I}}(ins(L, |L|+1, *), |L|+2, \alpha-1)$;               // Rule 9
```

2. Pevzner, P. A. & Sze, S.-H. Combinatorial Approaches to Finding Subtle Signals in DNA Sequences. In *ISMB*, vol. 8, 269–278 (2000).

3. Nicolae, M. & Rajasekaran, S. Efficient Sequential and Parallel Algorithms for Planted Motif Search. *BMC bioinformatics* **15**, 34 (2014).

4. Tanaka, S. Improved Exact Enumerative Algorithms for the Planted (l,d)-motif Search Problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* **11**, 361–374 (2014).

5. Karlin, S., Ost, F. & Blaisdell, B. E. Patterns in DNA and Amino Acid Sequences and Their Statistical Significance. In Waterman, M. S. (ed.) *Mathematical Methods for DNA Sequences* (CRC Press Inc. Boca Raton, FL, USA, 1989).

6. Rocke, E. & Tompa, M. An Algorithm for Finding Novel Gapped Motifs in DNA Sequences. In *Proceedings of the Second Annual International Conference on Computational Molecular Biology*, 228–233 (ACM, 1998).

7. Sagot, M.-F. Spelling Approximate Repeated or Common Motifs using a Suffix Tree. In *LATIN'98: Theoretical Informatics*, 374–390 (Springer, 1998).

8. Lanctot, J. K., Li, M., Ma, B., Wang, S. & Zhang, L. Distinguishing string selection problems. *Information and Computation* **185**, 41–55 (2003).

9. Adebiyi, E. & Kaufmann, M. Extracting Common Motifs under the Levenshtein Measure: Theory and Experimentation. *Algorithms in Bioinformatics* 140–156 (2002).

10. Rajasekaran, S. *et al.* High-performance Exact Algorithms for Motif Search. *Journal of Clinical Monitoring and Computing* **19**, 319–328 (2005).

11. Pathak, S., Rajasekaran, S. & Nicolae, M. EMS1: An Elegant Algorithm for Edit Distance Based Motif Search. *International Journal of Foundations of Computer Science* **24**, 473–486 (2013).

12. Yang, X., Chockalingam, S. P. & Aluru, S. A Survey of Error Correction Methods for Next Generation Sequencing. *Briefings in Bioinformatics* **14**, 56–66 (2013).

13. Knuth, D. E. *The Art of Computer Programming, Volume 4, Generating All Tuples and Permutations, Fascicle 2* (Addison Wesley, 2005).

Table 1. Conditions for skipping compact motif $L = \langle M, S_{j,k}, T\rangle$

Rule	**Conditions** (in all rules $t \geq 0$)
1	$(j+k \leq m) \wedge \langle j, D\rangle \in T$
2	$\{\langle j+t, D\rangle, \langle j+t+1, R\rangle\} \subseteq T$
3	$(j+k \leq m) \wedge \{\langle j, R\rangle, \langle j+1, R\rangle, \ldots, \langle j+t, R\rangle, \langle j+t+1, D\rangle\} \subseteq T$
4	$\{\langle j+t, D\rangle, \langle j+t, I\rangle\} \subseteq T$
5	$\{\langle j+t, D\rangle, \langle j+t+1, I\rangle\} \subseteq T$
6	$\{\langle j+t, R\rangle, \langle j+t, I\rangle\} \subseteq T$
7	$(j > 1) \wedge \langle j, I\rangle \in T$
8	$(j > 1) \wedge \{\langle j, R\rangle, \langle j+1, R\rangle, \ldots, \langle j+t, R\rangle, \langle j+t+1, I\rangle\} \subseteq T$
9	$(j+k \leq m) \wedge \langle j+k, I\rangle \in T$

Table 2. Possible alignments at position p where at least one of the pairs O_p, O'_p and N_p, N'_p differs

Case	O_p	N_p	O'_p	N'_p	Comments
1	σ	σ	σ	$*$	Not possible as there is no other character to match σ in N'
2	σ	σ	σ	$-$	Same as case 1
3	σ	$*$	σ	σ	Symmetric to case 1
4	σ	$*$	σ	$-$	Discussed in details in the text
5	σ	$-$	σ	σ	Symmetric to case 2
6	σ	$-$	σ	$*$	Symmetric to case 4
7	$-$	$*$	$-$	$-$	Discussed in details in the text
8	$-$	$-$	$-$	$*$	Symmetric to case 7

Table 3. Comparison between EMS1 and EMS2 on challenging instances. Time is in seconds (s), minutes (m) or hours (h). An empty cell implies the algorithm did not complete in the stipulated 72 hours. †estimated value.

	Run Time		**Memory Usage**	
Instance	EMS1	EMS2	EMS1	EMS2
(9,2)	11.9s	2.6s	3 MB	27 MB
(11,3)	35.8m	1.8m	91 MB	474 MB
(13,4)	-	1.2h	2,760[†] MB	7,264 MB

Acknowledgments

This work has been supported in part by the NIH grant R01-LM010101.

Author contributions statement

S.P. and S.R. designed the algorithms, S.P. conducted the experiments, S.P. and S.R. analysed the results, wrote and reviewed the manuscript.

Additional information

The authors declare no competing financial interest.

Expected Number of Spurious Motifs

Let L be a substring of length $l-q$, $0 \leq q \leq d$. If L is an occurrence of a fixed motif M with δ deletions, α insertions and β substitutions then $\alpha = q+\delta, 0 \leq \delta \leq (d-q)/2$, and the number of such d-neighbours of L is

$$N'(\delta,\beta) = \binom{l-q}{\delta} |\Sigma|^{q+\delta} \binom{l-q-\delta+1}{q+\delta} \binom{l-q-\delta}{\beta}. \tag{19}$$

and the probability that M is such a neighbour of L is

$$\left(\frac{|\Sigma|-1}{|\Sigma|}\right)^{\beta} \left(\frac{1}{|\Sigma|}\right)^{l-q-\delta-\beta}. \tag{20}$$

Table 4. Expected Number of Spurious Motifs in Random Instances for $n = 20, m = 600$

l	$d=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
5	0.0	1023.9	1024.0	1024.0	$>10k$									
6	0.0	1566.2	4096.0	4096.0	$>10k$	$>10k$								
7	0.0	0.1	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$							
8	0.0	0.0	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$						
9	0.0	0.0	1.8	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$					
10	0.0	0.0	0.0	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$				
11	0.0	0.0	0.0	31.8	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$			
12	0.0	0.0	0.0	0.0	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$		
13	0.0	0.0	0.0	0.0	509.7	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	
14	0.0	0.0	0.0	0.0	0.0	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$
15	0.0	0.0	0.0	0.0	0.0	8016.7	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$
16	0.0	0.0	0.0	0.0	0.0	0.0	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$
17	0.0	0.0	0.0	0.0	0.0	0.0	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$	$>10k$

Thus the probability that L is an occurrence of M is

$$= \sum_{\delta=0}^{\frac{d-q}{2}} \sum_{\beta=0}^{d-q-2\delta} N'(\delta,\beta) \left(\frac{|\Sigma-1|}{|\Sigma|}\right)^{\beta} \left(\frac{1}{|\Sigma|}\right)^{l-q-\delta-\beta}. \tag{21}$$

Similarly, if L is a substring of length $l+q$, $0 < q \le d$ the probability that L is an occurrence of M is

$$= \sum_{\delta=q}^{\frac{d+q}{2}} \sum_{\beta=0}^{d+q-2\delta} N''(\delta,\beta) \left(\frac{|\Sigma-1|}{|\Sigma|}\right)^{\beta} \left(\frac{1}{|\Sigma|}\right)^{l+q-\delta-\beta} \tag{22}$$

where

$$N''(\delta,\beta) = \binom{l+q}{\delta} |\Sigma|^{\delta-q} \binom{l+q-\delta+1}{\delta-q} \binom{l+q-\delta}{\beta}. \tag{23}$$

Combining the two cases, for L of length $l+q$, $-d \le q \le d$, the probability that L is an occurrence of M is $P=$

$$\sum_{\delta=\max\{0,q\}}^{\frac{d+q}{2}} \sum_{\beta=0}^{d+q-2\delta} N(\delta,\beta) \left(\frac{|\Sigma-1|}{|\Sigma|}\right)^{\beta} \left(\frac{1}{|\Sigma|}\right)^{l+q-\delta-\beta} \tag{24}$$

where

$$N(\delta,\beta) = \binom{l+q}{\delta} |\Sigma|^{\delta-q} \binom{l+q-\delta+1}{\delta-q} \binom{l+q-\delta}{\beta}. \tag{25}$$

There could be $(m-l-q+1)$ number of $(l+q)$-mers of a string of length m. The probability that M does not occur in the string is

$$R = \Pi_{q=-d}^{d} (1-P)^{m-l-q+1}. \tag{26}$$

The probability that M occurs in each of the input string is $(1-R)^n$. Since M can be any arbitrary motif, the expected number of motifs of length l at an edit distance d is

$$= |\Sigma|^l (1-R)^n. \tag{27}$$