人工知能学会研究会資料
SIG-FPAI-B404-06

移動付き編集距離のオンラインパターンマッチング

Online Pattern Matching for Edit Distance with moves

高畠嘉将 [1*]　田部井靖生 [2]　坂本比呂志 [1]
Yoshimasa Takabatake[1]　Yasuo Tabei [1,2]　Hiroshi Sakamoto[1]

[1] 九州工業大学大学院情報工学府
[1] Graduate School of Computer Science and Systems Engineering,
Kyushu Institute of Technology, Japan
[2] PRESTO さきがけ-独立行政法人科学技術振興機構
[2] PRESTO,Japan Science and Technology Agency

Abstract: Edit distance with moves(EDM) is a string-to-string measure such that inclues substring moves in addiction to ordinal editing operations to turn one string to the other. EDM is applicable to error detections and suggesting keywords for searches. Online ESP(OESP) is a first online pattern matching for EDM. OESP builds a parse tree for a whole input text in an online manner and quickly comutes EDM in each position. However, the computaion of EDM of OESP only uses subtrees that generates a query length substring. The space for a parse tree is too large for the computation of EDM. Thus, we present a more space efficient OESP called improved OESP(OESP-I). OESP-I builds from a parse tree for i-th position query length string to a parse tree for i+1-th position query length string and quickly computes EDM. We show the time and space compreixity of OESP-I. Additionally, we show the approximate ratio of EDM of OESP-I.

1　はじめに

本研究では移動付き編集距離 (EDM) のオンラインパターンマッチングを可能とする Online ESP(OESP)[8] を改良した Improve Online ESP(OESP-I) を提案する．EDM とは文字列間の距離指標の一つである．それは通常の編集距離の編集操作である置換，削除，挿入に部分文字列の移動を加えた編集操作を用いて，ある文字列から別の文字列への変換を行うのに必要な最小の編集操作回数で表現される．

移動付き編集距離は通信路のエラー発見 [5]，タイプミスの発見 [3]，生物学的配列の進化的変化の発見 [4] や検索キーワードのサジェストに用いることができるが，文字列間の移動付き編集距離を計算することは NP-Complete であることが知られている [6].

したがって，近似 EDM を求めるオフラインの手法がいくつか提案された [6, 7, 1]．その中の一つに ESP[1] という手法がある．ESP と同じ方法でオンラインで EDM を計算する手法 Online ESP(OESP)[8] も提案され，ストリームテキストに対しても近似 EDM を高速に計算することが可能となった．

*連絡先： 九州工業大学
〒 820-8502　福岡県飯塚市川津 680-4
E-mail: takabatake@donald.ai.kyutech.ac.jp

OESP は入力文字列全体を導出する構文木をオンラインで構築しながら，各位置の EDM を計算する．しかしながら，OESP にはまだ領域に無駄がある．なぜなら，各位置の EDM を計算する際には各位置の長さがクエリ長の部分文字列から構築される部分木の情報のみしか使用していないのに，入力文字列全体の構文木を持っているからである．

そこで本研究ではクエリ長を q とすると，入力文字列の i 番目から $i+q$ 番目の文字列を導出する ESP の構文木から入力文字列の $i+1$ 番目から $i+1+q$ 番目の文字列を導出する ESP の構文木を構築するデータ構造とアルゴリズムを提案する．また，EDM の近似率はオフラインの ESP と同じであることを示した．結果を表 1 に示す．

2　準備

2.1　基本概念

Σ は入力テキストを構成する文字の有限集合とする．Σ の要素数を σ もしくは $|\Sigma|$ と表記する．Σ の文字から構成されるすべての列の集合を Σ^*，Σ 上の文字から構成される長さ t のすべての文字列の集合を Σ^t と表記

表 1: EDM のオンラインパターンマッチングを行った場合のオフラインの手法と OESP との比較. EDM の近似率の upper bound, 計算時間，領域を示す. ESP と OESP と OESP-I の領域はビット表現. N は入力長，q はクエリ長; σ は入力文字種類数; n は入力文字列全体を導出する生成規則の数; n_q は入力文字列上の長さ q の部分文字列を導出する中で最大の生成規則数;$\alpha \in (0,1]$ はハッシュテーブルの load factor; $\lg^*$ は lg の繰り返し回数; lg は $\log_2$.

	Appro. ratio	Time	Space	Algorithm
SNN [6]	$O(\lg q \lg^* q)$	$O((q^{O(1)} + q\text{polylog}(q))N)$	$O(q^{O(1)})$	Offline
Shapira and Storer [7]	$O(\lg q)$	$O(q^2 N)$	$O(q \lg q)$	Offline
ESP [1]	$O(\lg q \lg^* q)$	$O(Nq \lg^* q/\alpha)$	$q \lg \sigma$ $+n_q(\alpha+3)\lg(n_q+\sigma)+2n_q \lg q$	Offline
OESP	$O(\lg q \lg N)$	$O(\frac{N \lg N \lg n}{\alpha \lg \lg n})$	$n(\alpha+1)\lg(n+\sigma)$ $+n\lg(\alpha n)+5n+o(n)+2n_q \lg q$	Online
OESP-I	$O(\lg q \lg^* q)$	$O(\frac{N(\lg^* q)^2 \lg q}{\alpha})$	$2n_q \lg q + n_q(\alpha+5)\lg(n_q+\sigma)$	Online

する．$\mathcal{X}$ は $\Sigma \cap \mathcal{X} = \phi$ を満たす帰納的可算集合であり，$\Sigma \cup \mathcal{X}$ のすべての要素は全順序関係を持つ．本論文では $\Sigma \cup \mathcal{X}$ の文字からなる列を 文字列と呼ぶ．文字列 S の長さを $|S|$ と表記し，集合 C の濃度も同様に $|C|$ と表記する．連続する 2 または 3 文字をそれぞれ *digram*, *triple* と呼ぶ．文字列 $S = xyz$ の x と z はそれぞれ接頭辞，接尾辞といい，x, y, z はすべて S の部分文字列という．文字列 S の i 番目の文字は $S[i](1 \le i \le |S|)$ と表記する．S の i 番目から j 番目の文字列は $S[i, j]$ と表記する．本論文上では S の長さを N とする．ここで N はオンラインを仮定するので，可変長である．文字列 S に出現する文字種類数を $[S]$ と表記する．

2.2 文脈自由文法

文脈自由文法 *(CFG)* は $G = (\Sigma, V, D, X_S)$ の 4 つ組から構成される．V は $\mathcal{X}$ の有限部分集合，D は生成規則 $V \times (V \cup \Sigma)^*$ の有限部分集合，$X_S \in V$ は開始記号である．D は辞書とも呼び，V の変数は非終端記号と呼ぶ．$L(G)$ は G により X_S から導出される Σ^* の文字列の集合である．すべての $X \in \mathcal{X}$ に対して，$X \to \gamma \in D$ を一意に持つ CFG G は *admissible* であるという．G のサイズは $|G|$ と表記し，生成規則の右側の文字列の長さの合計である．本論文上ではすべての生成規則 $X \to \gamma$ の右側の長さ $|\gamma|$ は 2 もしくは 3 に限定する．

G の構文木は V 上の変数によってラベリングされた内部ノードと Σ 上の文字を葉ノードとする根付き順序木である．ここで G の構文木の葉のラベル列は入力文字列である．構文木上のすべての内部ノード $Z \in V$ は $Z \to XY$ または $Z \to WXY$ の形の生成規則に対応する．

2.3 辞書と逆引き辞書

辞書 D とは P を生成規則の集合とすると，$Z \to S \in P$ のとき，すべての与えられた $Z \in V$ から $S \in (\Sigma \cup V)^*$ にアクセスするデータ構造である．逆引き辞書 $D^{-1} : (\Sigma \cup V)^* \to V$ とは生成規則の右側の文字列から，左側の文字列への射影である．すなわち，D^{-1} は $Z \to S \in D$ の時，すべての与えられた S から Z を返す.

2.4 移動付き編集距離

文字列 S と Q の移動付き編集距離 *(EDM)*$d(S, Q)$ とは 2 つの文字列 S を Q に変換するのに最小の編集操作回数である．ここで定義される編集操作を以下に示す:

1. 挿入: S 中の位置 i に文字 a を挿入し，$S[1, i-1]aS[i, N]$ を生成する．
2. 削除: S 中の位置 i の文字を削除し，$S[1, i-1]S[i+1, N]$ を生成する．
3. 置換: S 中の位置 i の文字を a に置換し，$S[1, i-1]aS[i+1, N]$ を生成する．
4. 部分文字列の移動: S 中の位置 i から j までの部分文字列 $S[i, j]$ を削除し，位置 k に挿入し，$S[1, i-1]S[j+1, k-1]S[i, j]S[k, N]$ を生成する．

2.5 問題定義

問題 1(Online pattern matching for EDM)[8] ストリームテキストデータ $S \in \Sigma^*$，クエリ $Q \in \Sigma^*$，距離閾値 k が与えられて，$d(S[i, i+|Q|], Q) \le k$ となる位置 $i \in [1, N]$ をすべて探す.

Cormode and Muthukrishnan[2] は EDM を計算するためのオフラインアルゴリズムを提案した．この手法は ESP 木と呼ばれる特殊な構文木を構築して，近似的に EDM を計算する．

2.6 Edit sensitive parsing

我々の手法は Edit sensirive parsing(ESP)[2] に基づいているため，その手法について説明する．ESP は，(i) 入力文字列 $S \in \Sigma^*$ すべてを digram または trigram に分割し，それぞれ 2 木 ($Z \to XY$) または 3 木 ($Z \to WXY$) を構築し，構築された部分木の列の根の非終端記号列を入力文字列 S として，入力文字列の長さが 1 になるまでこの構築を繰り返し，最初の入力文字列 S を導出する構文木 (ESP 木) を構築する．(ii)(i) で構築した ESP 木の各非終端記号の出現回数を各要素とする特徴ベクトル V_S を生成する．(iii)$Q \in \Sigma^*$ に対しても同様に ESP 木を構築し，特徴ベクトル V_Q を生成する．(iv) 近似 EDM となる V_S と V_Q の L1 距離を計算する．

2.6.1 アルゴリズム

このアルゴリズムでの各 2 または 3 木の構築においては同じ digram もしくは trigram は同じ非終端記号で置き換えられるように辞書 D と逆引き辞書 D^{-1} を用いて，管理している．まず，入力文字列 S を以下の 3 つの文字列に分割する．

Type1: 連続文字.

Type2: 連続文字を含まない長さが $\lg^* N$ 以上の文字列.

Type3: Type1 と Type2 以外の文字列.

Type1 から 3 の部分文字列はそれぞれ以下のように digram または trigram に分割し，それぞれ 2 または 3 木を構築する．

Type1 と Type3: 以下のように Type1 または Type3 の部分文字列 s から左優先で 2 木を構築し，長さが奇数の場合は最後の trigram からは 3 木を構築する．

s の長さが偶数: $Z_i \to s[2i-1, 2i](i \in [1, |s|/2]))$ を 構築する．

s の長さが奇数: $Z_i \to s[2i-1, 2i](i \in [1, (|s|-1)/2-1])), Z_{(|s|-1)/2} \to s[|s|-2, |s|]$ を構築する．

Type2: alphabet reduction により digram または trigram に文字列を分割し，それぞれ 2 または 3 木を構築する．

Alphabet reduction: Type2 の部分文字列を digram または trigram に分割し，それぞれ 2 または 3 木を構築する手順である．まず，Type2 の部分文字列 s からラベル $L[i] = 2p + bit(p, i)(2 \le i \le |s|)$ を計算する．ここで p は $s[i]$ と $s[i-1]$ を二進数で表現し，最下位ビットから上位ビットへと見ていったときに最初に異なるビットの位置であり，$bit(p, i)$ は $s[i]$ の p 番目のビット値である．ここで最下位ビットの位置は 0 とする．この計算により L の文字種類数は $[L] \le 2\lg[s]$ となるので，生成された L を新たな入力文字列として，この計算を文字種類数が $\lg^* N$ 以下のラベル列 L^* になるまで繰り返す．L^* 上で $L^*[i]$ が極大値 $L^*[i] > max\{L^*[i-1], L^*[i+1]\}$ の時，位置 $S[i]$ を *landmark* と呼び，2 木 $Z \to S[i-1, i]$ を構築する．$L^*[i-1]$ と $L^*[i+1]$ が極大値でなく，$L^*[i]$ が極小値 $L^*[i] < min\{L^*[i-1], L^*[i+1]\}$ の時も位置 $S[i]$ を landmark と呼び，2 木 $Z \to S[i-1, i]$ を構築する．すべての landmark とその 2 木を構築した後で，残りの部分文字列に対しては Type1 と 3 と同様に左優先で 2 または 3 木を構築する．一回のラベル計算において，Type2 の文字列からは Type2 のラベル列が計算される．したがって，$[L^*] \le \lg^* N$ より，landmark は $\lg^* N$ 内に必ず一つは存在する．

構築された部分木の列の根ノードに対応する非終端記号を新たな入力文字列 S として，ここまでの操作を $|S| = 1$ となるまで繰り返し，最初の入力文字列 S を導出する構文木 (ESP 木) を構築する．

Q に対しても ESP 木を構築する．このとき，S の時と同じ辞書と逆引き辞書を用いて，S の ESP 木上と同じ digram と trigram から構築される非終端記号は同じ非終端記号名となるようにする．

次に S の ESP 木 T_S 上に存在する各非終端記号の出現回数を各次元とする特徴ベクトル V_S を構築する．$x \in V$ とすると，$V_S[x]$ は x の出現回数を表す．Q の ESP 木 T_Q に対しても同様に特徴ベクトル V_Q を構築する．

最後に V_S と V_Q の L1 距離 $||V_S - V_Q|| = \Sigma_{x \in T_S \cup T_Q} |V_S[x] - V_Q[x]|$ を計算する．これが EDM の近似になっており，定理が成り立つ．

定理 1. (Cormode and Muthukrishnan[2]) 文字列 Q と S，$N = max(|Q|, |S|)$ が与えられた時，$||V_Q - V_S|| \le O(\lg N \lg^* N) d(Q, S)$ が成り立つ．

また，以下の重要な補題および定理が示されている．

補題 2. (Cormode and Muthukrishnan[2])Type2 において位置 i に最も近い landmark は左 $\lg^* N + 5$ 文字と右 5 文字により決定される．

この補題は同じ文字列上に同じ Type2 の部分文字列があるとき，最初の landmark 以降の分割は同じになるということを述べている．

定理 3. (Cormode and Muthukrishnan[2]) 入力文字列 S から T_S を構築するのに $|S| \lg^* |S|$ 時間かかる．

この定理は逆引きを $O(1)$ とみなしている．本論文ではハッシュテーブルの load factor α を用いて，逆引きを $O(1/\alpha)$ 時間とみなすため，$O(1/\alpha|S|\lg^*|S|)$ 時間とする．

さらに Takabatake et al. [8] では入力文字列 $S \in \Sigma^*$ に対する ESP 木をオンラインで構築して，各位置の近似 EDM を高速に計算する手法が提案されている．この手法では，S 全体に対する構文木を構築しているが，各位置の EDM の近似計算をするために使っている情報は $S[i, i+|Q|]$ から構築される部分木の情報のみであり，領域に無駄がある．そこで本論文では，$S[i, i+|Q|]$ の ESP 木から $S[i+1, i+1+|Q|]$ とほぼ等価な ESP 木を高速に構築し，各位置の近似 EDM を高速に計算するデータ構造とアルゴリズムを提案する．

3 Improved Online ESP

本論文では省領域でかつ高速に ESP の近似 EDM をオンラインで計算するデータ構造とアルゴリズム Improved Online ESP(OESP-I) を提案する．また，その ESP 木がオフラインで構築した ESP 木とほぼ等価であることを示し，オフラインの ESP の近似 EDM と upper bound が同じであることを示す.

3.1 データ構造

提案するデータ構造には辞書 D，逆引き辞書 D^{-1}，入力文字列 $S[j, j+|Q|]$ とクエリ Q の特徴ベクトルを保存するデータ構造 V_S と V_Q ，非終端記号を保存するスタック ST を用いる．D は $X_l \to X_iX_jX_k$ という生成規則の l を添え字 (i, j, k) の三つ組みをデータとする三重配列で表現する．生成規則が $X_l \to X_iX_j$ のように右側の長さが 2 の時は (i, j, d) のように 3 番目に $X_d \notin D$ となる dummy code d を入れておく．

D のサイズは生成規則の数を n とすると，$3n\lg(n+\sigma)$ である．一回のアクセスを $O(1)$ で行うことが可能である．

D^{-1} にはハッシュテーブルを用いる．ハッシュテーブルの load factor を α とすると，領域 $n(1+\alpha)\lg(n+\sigma)$ で一回の逆引きを $O(1/\alpha)$ 時間で計算可能である．

V_S と V_Q は配列であり，X_i を非終端記号とすると，$V_S[i](V_Q[i])$ には $T_{S[j,j+|Q|]}(T_Q)$ 上での X_i の出現回数が保存されている．V_S と V_Q のサイズは片方が $n\lg q(q = |Q|)$ であり，両方合計して，$2n\lg q$ である．各次元の出現回数には $O(1)$ 時間でアクセスすることが可能である．

スタック ST には現在使用されていない非終端記号 X_i の i が保存されている．ST は領域の削減のために使用する．ST の長さは最大で n であるので，そのサイズは $n\lg(n+\sigma)$ である．

したがって，全体のサイズは以下のようになる．

定理 4. クエリの長さを q，生成規則の数を n，文字種類数を σ，ハッシュテーブルの load factor を α とすると，OESP-I のデータ構造のサイズは $2n\lg q + n(\alpha+5)\lg(n+\sigma)$ である．

3.2 アルゴリズム

まず，クエリ Q と入力文字列 $S[1, 1+|Q|]$ の ESP 木 $T_{S[1,1+|Q|]}$ と T_Q を構築し，特徴ベクトル V_S と V_Q を計算し，位置 1 の L1 距離 $L_1[1]$ を計算する．次に $S[i, i+|Q|]$ の ESP 木が構築されており，$L_1[i]$ までが計算されているとする．その時，$T_{S[i,i+|Q|]}$ から $S[i]$ を削除し，$S[i+1+|Q|]$ を追加して $T_{S[i+1,i+1+|Q|]}$ を以下のようにして構築する．

最初に $T_{S[i,i+|Q|]}$ を図 1 のように構文木が左右ともに階段状になるようにノードを削除する．$L_1[i+1] = L_1[i]$ とする．まず左側を階段状に削除する．高さ 0 の文字列の先頭 $S_0[i, i+t_0]$ に対応するノードそれぞれから根ノードまでのパスにあるノードを削除する．高さ j でも同様にして残っている非終端記号列 S_j の先頭 $S_j[1, 1+t_j]$ に対応するノードそれぞれから根ノードまでのパスにあるノードを削除する．この削除は高さの低い方から順に行う．各深さの t_j の決定は残っている終端および非終端記号列の先頭が以下のように ESP のどの Type に属すかで決定する．

先頭が Type1: $S_j[i]$ から 2 木が構築されていた場合は隣りの深さ j 部分木が 2 木の場合は $t_j = 3$ とし，隣りの部分木が 3 木の場合は $t_j = 4$ とする．$S_j[i]$ から 3 木が構築されていた場合は $t_j = 2$ とする．

先頭が Type2: $S_j[i+1, i+1+|Q|]$ 上の最初の landmark の位置 l とすると，$t_j = l - i$ とする．

先頭が Type3: $S_j[i]$ を含む Type3 の部分文字列の長さを k とすると，$t_j = k$ とする．

次に構文木の右側を階段状になるようにノードを削除する．左の時と同様にして，深さの低い方から順に各深さの残っている終端および非終端記号列の末尾が ESP のどの Type に属すかによって，根ノードから各深さのノードまでのパス上にあるノードを削除するかを決定する．深さ j で残っているノードに対応する終端および非終端記号列 S_j とし，S_j の末尾 $S_j[|S_j|-s_j, |S_j|]$ に対応するノードそれぞれから根ノードまでのパスが削除されるとすると，s_j は以下のように決定される．

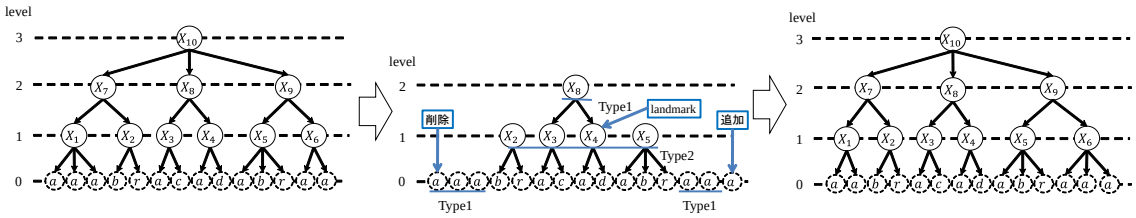

図 1: $S[i, i+|Q|]$ の ESP 木から $S[i+1, i+1+|Q|]$ の ESP 木の構築の例．S は入力文字列，Q はクエリ．

末尾が Type1 と Type3: $S_j[|S_j|]$ から 2 木が構築されている場合 $s_j = 1$ または 3 木が構築されている場合 $s_j = 2$ とする．

末尾が Type2: $S_j[|S_j|]$ にもっとも近い landmark の位置を l とすると，$s_j = |S_j| - l$

この各ノードの削除操作を行うときには同時に特徴ベクトル V_S の更新と L1 距離の更新も以下のように行う．非終端記号 X_i が削除されるとき，$L_1[i{+}1] := L_1[i{+}1] - |V_S[i] - V_Q[i]|$ とし，特徴ベクトルを $V_S[i] := V_S[i] - 1$ と更新し，$L_1[i+1] := L_1[i+1] + |V_S[i] - V_Q[i]|$ とする．さらに $V_S[i] = 0$ かつ $V_Q[i] = 0$ となった場合は D および D^{-1} から生成規則 X_i を削除し，ST に i を push しておく．

次に階段状にノードが削除された構文木の先頭から $S[i]$ を削除し，末尾に $S[i+1+|Q|]$ を追加する．階段状に削られた部分に対して，ESP を各深さごとにもう一度行い，$T_{S[i+1,i+1+|Q|]}$ を構築する．このとき，2 または 3 木を構築するときには非終端記号の名前付けは ST が空の場合は辞書 D に無い新しい名前をつける．ST が空でない場合は ST から pop した k を新しい X_k として扱う．さらに同時に L1 距離と特徴ベクトルの更新を行う．X_j が構築されたときには，$L_1[i+1] := L_1[i{+}1] - |V_S[j] - V_S[j]|$ とし，$V_S[j] := V_S[j]{+}1$ とベクトルを更新し，$L_1[i{+}1] := L_1[i{+}1] + |V_S[j] - V_S[j]|$ とする．こうして一つの ESP 木を構築することで，$L_1[i{+}1]$ も同時に計算することが可能である．

3.3 解析

補題 5. q をクエリ長，α をハッシュテーブルの load factor とすると，位置 i から $i+1$ の $L1$ 距離を計算する時間は $O(1/\alpha \lg^* q \lg q)$ 時間である．

証明. 特徴ベクトルの削除操作および追加操作による L1 距離の一回の更新操作は $O(1)$ 時間である．入力文字列の先頭の削除操作および末尾追加操作によって ESP 木上の深さ i で導出される文字列は Type1 は定数個，Type2,Type3 は最大で $\lg^* q$ であるので，削除されるノード数は $O(\lg^* q \lg q)$ である．したがって，削除操作には $t_1 = O(\lg^* q \lg q)$ 時間かかる．辞書の逆引きには $O(1\alpha)$ 時間かかり，landmark を計算するためには $\lg^* q$ 時間かかるので，先頭の削除および末尾追加操作における 2 または 3 木の構築操作は全体として,$t_2 = O(1/\alpha(\lg^* q)^2 \lg q)$ 時間かかる．したがって，$L1[i]$ から $L1[i{+}1]$ を計算するのに $t_1{+}t_2 = O(1/\alpha(\lg^* q)^2 \lg q)$ 時間かかる． □

定理 6. 入力長を N，クエリ長を q，ハッシュテーブルの load factor を α とすると，OESP-I の Online patterm matching for EDM の計算時間は $O(1/\alpha(N(\lg^* q)^2 \lg q))$ 時間である．

証明. Q および $S[1, 1{+}|Q|]$ の ESP 木および L1 距離を計算するのに $t_1 = O(1/\alpha q \lg^* q)$ 時間かかる．補題 5 より，各位置の L1 距離の計算には $t_2 = O(1/\alpha(\lg^* q)^2 \lg q)$ 時間かかるので，全体としては，$t_1 + N * t_2 = O(1/\alpha (N(\lg^* q)^2 \lg q))$ 時間かかる． □

定理 7. 入力文字列を S，クエリを Q，クエリ長を q，V_S を $S[i, i{+}|Q|]$ の特徴ベクトル，V_Q を Q の特徴ベクトルとすると，$||V_S - V_Q|| = O(\lg^* q \lg q) d(S[i, i+|Q|], Q)$ である．

証明. オフラインの ESP で構築される $T_{S[i,i+|Q|]}$ との違いは非終端記号を削除し導出した文字列が Type1 の時のみである．なぜなら Type1 の場合はオフラインでは左優先で 2 木を作るのに対して，先頭に 3 木を作ることになることがあるからである．しかしながら，この違いは ESP 木の各高さで定数個しかないので，これによる L1 距離のずれは $O(\lg q)$ である．また Type2 の場合は，最初の landmark 以降の置き換えは一致するので，landmark までを一度導出して，構文木を作り直しているので分割はオフラインの時と一致する．Type3 の場合は次の Type の文字列以降の分割は一致するので，すべて導出し，作りなおしているので，分割はオフラインの時と一致する． したがって，オフラインの ESP との L1 距離の違いは $O(\lg q)$ であり，定理 1 より，$||V_S - V_Q|| = O(\lg^* q \lg q) d(S[i, i+|Q|], Q)$ である． □

4 おわりに

Online ESP より近似 EDM の計算をさらに作業領域を減らす手法 Improved Online ESP を提案した．また，EMD に対する upper bound をオフラインの ESP と同じで計算することを示した．今後実装し，実験を行い，その実用性を確かめる．

参考文献

[1] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *TALG*, Vol. 3, pp. 2:1–2:19, 2007.

[2] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. *ACM Trans. Algor.*, Vol. 3, No. 1, p. Article 2, 2007.

[3] M. Crochemore and W. Rytter. *Text Algorithms*. Oxford University Press, 1994.

[4] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

[5] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, Vol. 10, pp. 707–710, 1996.

[6] S Muthukrishnan and S. C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. In *Proc. of STOC*, pp. 416–424, 2000.

[7] D. Shapira and J. A. Storer. Edit distance with move operations. *JDA*, Vol. 5, pp. 380–392, 2007.

[8] Y. Takabatake, Y. Tabei, and H. Sakamoto. Online pattern matching for string edit distance with moves. In *SPIRE2014, to appear*, 2014.