人工知能学会研究会資料
SIG-FPAI-B501-03

siEDM:移動付き編集距離の為の効率的な索引

siEDM:an efficient string index and search algorithm for edit distance with moves

高畠嘉将[1] 中島健太[1*] 田部井靖生[2] 坂本比呂志[1]
Yoshimasa Takabatake[1] Kenta Nakashima[1] Yasuo Tabei[1] Hiroshi Sakamoto[1]

[1] 九州工業大学 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: We propose an index and search algorithm based on the edit distance with moves. *Edit distance with moves (EDM)* is a distance measure between two strings, including substring moves in addition to edit distance to turn one string to the other. It is known to computing exact EDM is hard.Thus, an index structure and a search algorithm with EDM doesn't exist. We show the first algorithm named *string index for edit distance with moves (siEDM)* for indexing and searching strings with an approximate EDM. Using *edit sensitive parsing (ESP)*, our method builds an index structure and prunes the search space for searching query by proving a bound of the approximate EDM. This bound for fast query searching is equal to the bound of ESP.

1 はじめに

本稿では，移動付き編集距離に基く近似検索を可能にする為の索引と検索アルゴリズムの提案を行う．ソースコードのリポジトリや Wikipedia の編集履歴等，所謂繰り返しの多いテキストの文書集合が一般に数多く存在する．例えば生物データベースには大量のヒトゲノムの情報が格納されているが，ゲノム同士の遺伝的差異は 0.1% 以下である．従って，大規模な繰り返しの多いテキスト集合に対して処理可能なアルゴリズムが必要とされている．現在の所，繰り返しの多いテキストを含む巨大なデータベースを検索する為の一般的な手法は，索引の構築である．大規模データに対する索引構築・検索アルゴリズムとして現在までに，ESP-index[4], SLP-index[1]，LZ-index[2] 等の複数の手法が提案されている．これらのアルゴリズムは高速クエリ検索を可能にするが，検索方法が完全一致検索のみに制限されている．移動付き編集距離とは文字列間の類似度の一種であり，片方の文字列を他方に変換する際に挿入・削除・置換・部分文字列移動の 4 つの操作の最小回数として定義されている．移動付き編集距離は，文書のタイプミスや DNA 配列の進化的変化等を含むエラー訂正に利用することができる．しかし，移動付き編集距離の正確な値を求めることは困難であることが知られており [5]，近似値を求めるアルゴリズムが提案されてきた．その 1 つに edit sensitive parsing(ESP)[3] という近似率 $O(\lg N \lg^* N)$，時間計算量 $O(N \lg^* N)$ で動く近似アルゴリズムが存在する．ここで N は入力長である．これまで，移動付き編集距離に基くテキスト検索と索引構築アルゴリズムは存在していなかった．今回，本研究では siEDM と呼ぶ大規模データに対する効率的な索引とそれを用いた移動付き編集距離の為の検索アルゴリズムを提案し実験を行った．

*連絡先： 九州工業大学大学院 情報工学府
〒 820-8502 福岡県飯塚市川津 680 番 4
E-mail: k_nakashima@donald.ai.kyutech.ac.jp

2 準備

文字列 S の長さを $|S|$，集合 C の基数を $|C|$ と書く．文字の有限集合を Σ とおき，Σ の要素からなる文字列全体の集合を Σ^* と定義する．$S[i]$, $S[i, j]$ をそれぞれ文字列 S の i 番目の文字と i から j 番目までの部分文字列とする．本稿中では，対数の表記は $\lg = \log_2$, $\lg^{(1)} u = \lg u, \lg^{(i+1)} u = \lg \lg^{(i)} u$ と表記する．また，$\lg^* u$ は $\lg^* u = \{min\{i | \lg^{(i)} u \leq 1\}\}$ を表す．文字列 S と正整数 q に対し，$pre(S, q), suf(S, q)$ をそれぞれ S の長さ q の接頭辞と接尾辞とする．

2.1 Straight-line program(SLP)

文脈自由文法 (CFG) は 4 つ組 $G = (\Sigma, V, D, X_s)$ として表される．V は変数の有限集合，D は生成規則の集合，X_s は開始記号である．X_i から導出可能な文字列を $val(X_i)$ と書き，変数 X_i の左右の子をそれぞれ $X_{l(i)}, X_{r(i)}$ と表記する．S を文法圧縮するということは，即ち S のみを導出する CFG G を生成することである．以降では，CFG のサイズを変数の数とし $n = |V|$ とする．G の parse tree は次の 3 つの条件を満たす 2 分木である．(i) 内部ノードには V 中の変数が割り当てられる．(ii) 葉ノードには Σ 中のシンボルが割り当てられる．(iii) 葉のラベルの並びは入力文字列 S と等しい．parse tree において，任意の内部変数 Z と生成規則 $Z \to XY$ が対応し，Z は左右の子として X, Y を持つ．ここで *Straight-line program(SLP)* は任意の生成規則 $X_k \to X_i X_j$ が $1 \leq i, j < k$ を満たす文法圧縮として定義される．

2.2 rank/select 辞書

rank/select 辞書とは以下の操作をサポートしているビット列の事である．

1. $rank_c(B, i)$ $B[0, i]$ 中の $c \in 0, 1$ の数を返す操作.
2. $select_c(B, i)$ $B[0, i]$ 中の i 番目に出現した $c \in 0, 1$ の位置を返す.
3. $access(B, i)$ B 中の i 番目のビットを返す.

$O(1)$ 時間で rank/select 操作を実現するために必要なデータサイズは $|B| + o(B)$[bit] である．GMR[6] は，巨大なアルファベットの為の rank/select 辞書であり，$(\Sigma \cup V)^*$ 内の文字列に対して rank，select，access をサポートしている．GMR では $(n+\sigma)\lg(n+\sigma) + o((n+\sigma)\lg(n+\sigma))$[bit] の領域で，rank と access を $O(\lg\lg(n+\sigma))$，select を $O(1)$ 時間で実行可能である．ここで，σ はアルファベットサイズである．

3 問題定義

本研究における問題を定義する為，はじめに移動付き編集距離 (EDM) について説明する．EDM $d(S, Q)$ とは，ある文字列 S から別の文字列 Q へ変換する際に以下の 4 つの編集操作の最小回数として定義される文字列間の距離のことである．

3.1 移動付き編集距離 (EDM)

1. **挿入:** 文字列の任意の場所に文字を挿入する操作.

$$abcd \to aba\underline{c}cd$$

2. **削除:** 文字列の任意の場所の文字を取り除く操作.

$$aba\underline{c}d \to abad$$

3. **置換:** 文字列中の任意の場所の文字を別の文字に置き換える操作. $aba\underline{d} \to ab\underline{c}d$
4. **部分文字列移動:** 文字列中の部分文字列を別の場所に移動する操作. $aaaa\underline{bbbb} \to \underline{bbbb}aaaa$

ここでは移動付き編集距離に基づく近似検索を以下のように設定する.

- **入力**

  テキスト $S \in \Sigma^*$，検索クエリ $Q \in \Sigma^*$，
  距離閾値 $\tau \geq 0$

- **出力**

  S の部分文字列 $S[i, i + |Q| - 1]$ と Q の移動付き編集距離 $d(S[i, i + |S| - 1], Q) \leq \tau$ となる全て部分文字列のポジション $i \in [1, |S|]$

4 Edit Sensitive Parsing(ESP)

Edit Sensitive Parsing(ESP)[3] とは 2 つの文字列 R, S の近似移動付き編集距離 $d(R, S)$ を計算するために提案された文字列の parsing algorithm である．
1. で述べたように，文字列間の移動付き編集距離を求めることは困難な為，ESP を用いた距離計算では近似解を求めることで計算を行っている．

4.1 ESP algorithm

ESP では長さ N の文字列 S から ESP 木と呼ばれる構文木を構築する [3]．この木は全 2 分木として表現できる [4]．S を以下の 3 つの Type の部分文字列の列とみなして長さ 2 または 3 文字毎に分割を行う.

Type1 長さが 2 以上の同一文字による連続文字列.

Type2 同一文字による連続文字列を含まない，長さ $\lg^* N$ を超える文字列.

Type3 Type1，Type2 共に当てはまらない文字列．即ち同一文字による連続文字列を含まない長さ $\lg^* N$ 以下の文字列である.

Type1，Type3 に関しては左優先で 2 文字ずつをペアにして置き換えを行っていく．長さが偶数の場合は，XY から $t_i \to XY$ が，長さ奇数の場合は最後の 3 文字を XYZ から $t_i \to Xt'$，$t' \to YZ$ のように 2-2 木を構築する．Type2 の文字列に関しては alphabet reduction[3] という方法で分割を行っていく．

ESP による分割の例として図 1を載せる．図 1は入力テキストに対し 1 段 ESP を適用した状態である．文字列 $S = abababbbab$ に対して $ACBDA$ という新たな文字列が得られている．
図 1のように葉を level0 として i 回 ESP を繰り返した時のノードを level i のノードと呼ぶ．このアルゴリズムによって出来た ESP 木を用いることで移動付き編集距離の近似解である L_1 距離を求める．

4.2 ESP 木からの近似 EDM 計算方法

L_1 距離とは以下の式によって定義される 2 つの文字列 R, S 間の距離である．

$$||F(S) - F(Q)||_1 = \sum_{e \in T(S) \cup T(Q)} |F(S)[e] - F(Q)[e]|$$

ここで，$T(S)$ を文字列 S から作られる ESP 木 T_S の全ノードの集合とする．$F(S)$ は $T(S)$ の各要素の ESP 木内での出現頻度を保存している非負整数ベクトルである．$F(S)$ を S の特徴ベクトルと呼ぶ．$F(S)[e]$ は $F(S)$ 中の要素 e の ESP 木内での出現回数を表している．L_1 距離と移動付き編集距離の関係について，定理 4.1が成り立つ．

定理 4.1 (Cormode and Muthukrishnan [3]) 文字列 Q，S，$m = MAX(|Q|, |S|)$ が与えられた時，以下の式が成り立つ．

$$d(S, Q) \leq 2||F(S) - F(Q)||_1 = O(\lg m \lg^* m) d(S, Q)$$

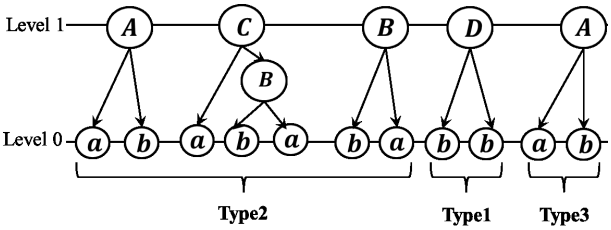

図 1: 文字列 $S = abababbbab$ に対する ESP の分割例

5 提案手法

5.1 ESP 木からの索引構築

ここでは，ESP 木から近似検索のための索引の構築方法について述べる．概略としては，元のテキストを表す生成規則を昇順にソートしラベルを付け替え符号化．その後，各ノードに対し特徴ベクトルと長さベクトルを生成する．

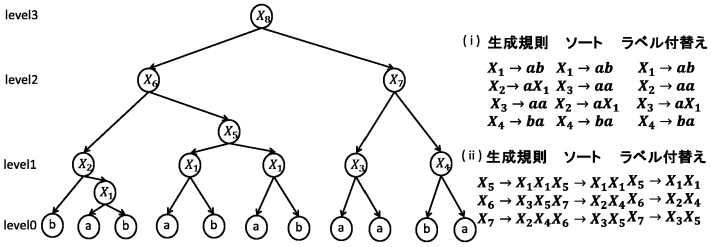

図 2: 文字列 $S = bababababaaba$ に対する ESP 木と符号化の為のソート

ESP 木のエンコード方法 siEDM では索引構築のために ESP 木に対し符号化を行う．始めに，level0 と次の level の生成規則をソートする．左の子の昇順にソートし親ノードのラベルを付け替える．この操作を次の level に対しても繰り返し行ってラベルを付け替えていく．図 2に例を示す．図では左側の ESP 木のノードに対してソートを行っている．(i) で level0 と level1 の変数についてソート，(ii) で次の level をソートしている．符号化された ESP 木のサイズは左の生成規則の表現にギャップエンコーディングと単身符号を用いることで $2n$[bit]，右の生成規則の表現に GMR[6] を用いることで合計 $(n+\sigma)\lg(n+\sigma) + 2(n+\sigma) + o((n+\sigma)\lg(n+\sigma))$[bit] で表現可能である．

符号化された ESP 木上での操作 符号化された ESP 木上では $LeftChild, RightChild, LeftParents, RightParents$ という 4 つの木への操作をサポートしている．$LeftChild(X_k)$ は X_k の左の子 $X_{l(k)}$ を返す操作であり，$O(1)$ で行える．$RightChild(X_k)$ は右の子を返す操作で $O(\lg\lg(n+\sigma))$ で行える．$LeftParents(X_k), RightParents(X_k)$ はそれぞれ X_k が左右の子にあるときの親ノードを返す関数で，$LeftParents(X_k) = \{X_i \in V; X_i \to X_kX_j, \forall X_j \in (\Sigma \cup V)\}, RightParents(X_k) = \{X_i \in V; X_i \to X_jX_k, \forall X_j \in (\Sigma \cup V)\}$ と表すことができる．かかる時間は，$LeftParents$ が $O(|LeftParents(X_k)|)$，$RightParents$ が $O(|RightParents(X_k)|)$ である．

その他データ構造 siEDM では高速なクエリ検索のために，ESP 木の各ノードに対しそのノードの ESP 木中の出現頻度を記録した特徴ベクトルを用意する．以降では，ノード X_i の特徴ベクトルを $F(X_i)$ と表記する．全てのノードの数を n とした時，全てのノードに対し記録したこの特徴ベクトルのサイズは合計 $n^2 \lg |S|$[bit] となる．また，X_i 以下の部分木に一度以上出現するノードの集合を $T(X_i) = \{e \in (V \cup \Sigma); F(X_i)(e)\}$ と表記する．さらに，別のデータ構造として各ノードを展開した時の部分文字列の長さを保存した長さベクトル $L(X_i)$ も用意する．$L(X_i)$ のデータサイズは $n \lg |S|$[bit] である．図 3に特徴ベクトル，長さベクトルの例を示している．図は図 2の ESP 木に対する 2 つのベクトルの例となっている．

$$
\begin{array}{l}
\qquad\quad a\ b\ X_1 X_2 X_3 X_4\ X_5 X_6 X_7 X_8 \\
F(X_1) = (1,1,1,0,0,0,0,0,0,0) \\
F(X_2) = (2,0,0,1,0,0,0,0,0,0) \\
F(X_3) = (1,2,0,0,1,0,0,0,0,0) \\
F(X_4) = (1,1,0,0,0,1,0,0,0,0) \\
F(X_5) = (2,2,2,0,0,0,1,0,0,0) \\
F(X_6) = (3,1,0,1,0,1,0,1,0,0) \\
F(X_7) = (3,4,2,0,1,0,0,0,1,0) \\
F(X_8) = (6,5,2,1,1,1,0,1,1,1)
\end{array}
\qquad
\begin{array}{l}
L(X_1) = 2 \\
L(X_2) = 2 \\
L(X_3) = 3 \\
L(X_4) = 2 \\
L(X_5) = 4 \\
L(X_6) = 4 \\
L(X_7) = 7 \\
L(X_8) = 11
\end{array}
$$

図 3: 文字列 $S = bababababaaba$ に対する ESP 木の特徴ベクトルと長さベクトル

以上から，siEDM の索引にかかる全データサイズは，$n(n+1)\lg|S| + (n+\sigma)\lg(n+\sigma) + 2(n+\sigma) + o((n+\sigma)\lg(n+\sigma))$ [bit] である．

5.2 検索アルゴリズム

ここでは構築した索引を使って EDM に基く近似検索を実現する為の検索アルゴリズムの説明を行う．アルゴリズムの流れとしては，1:候補地の抽出，2:L_1 距離の計算，3:該当するポジションの計算，に分けて行う．基本的なアイディアとしては，入力文字列 S から作られる符号化された ESP 木に対し全ての q-gram を計算することによって検索スペースを抑える．変数 $X_i \in V$ に大使 q-gram は以下のような 2 つの部分文字列に分解できる．(i)$X_{l(i)}$ の最右子孫の長さ $(q-k)$ の suffix，(ii)$X_{r(i)}$ の最左子孫から長さ k の prefix．これを $k = 1, 2, \cdots, (q-1)$ まで行った時，X_i にある全ての q-gram を見つけることができる．$itv(X_i) = \{suf(val(X_{l(i)}), q-k), pre(val(X_{r(i)}), k);$
$k = 1, 2, \cdots, (q-1)\}$ を X_I の全ての q-gram の集合とする．この時，$itv(S) = itv(X_1) \cup itv(X_2) \cup \cdots \cup itv(X_n)$ であり，$itv(S)$ は S に出てくる全ての q-gram を含んでいる．更に効率的に検索するために，検索スペースを枝刈りする為の L_1 距離に関する下限を示す．

定理 5.1 $F(S), F(Q)$ を文字列 S, Q の特徴ベクトルとする．任意の変数 $X_i \in T(S)$ に対し，以下の式を定義する．

$$\mu(X_i) = \sum_{e \notin T(Q)} F(X_i)(e)$$

この時，不等式 $||F(S) - F(Q)|| \geq \mu(X_i)$ が成り立つ．

証明. $F(S), F(Q)$ の L_1 距離は次の 4 つのグループに分類できる．(i)$F(S), S(Q)$ の要素が両方とも 0 ではない．(ii) $F(S), F(Q)$ の要素が両方とも 0．(iii)$F(S)$ が 0，$F(Q)$ が 0 以外のとき．(iv)$F(S)$ が 0 以外，$F(Q)$ が 0 のとき．
(iv) のみによって構成される条件の場合，$\sum_{e \notin T(Q)} F(S)(e)$ と表すことができ，L_1 距離の下限と一致する．
従って，$||F(S) - F(Q)|| \geq \sum_{e \notin T(Q)} F(S)(e)$ である． □

また，$\mu(X_i)$ には以下のような単調性がある．

定理 5.2 $X_i \in V$ の任意の子孫 $X_k \in V$ に対し，不等式 $\mu(X_k) \geq \mu(X_i)$ が成り立つ．

証明. 各ノードの要素数を比べた時，任意の $F(_k)$ は $F(X_i)$ 以下の要素数しか持たない．従って，不等式が成り立つ． □

5.3 候補地発見

定理 5.1,5.2を用いることで，候補地発見に関する以下のような $|Q|$-gram の最適化問題を解くことができるようになる．

$$\min_{\{x_{i_1}, x_{i_2}, , x_{i_m}\}} m \text{ s.t.}$$

$$|val(X_{i_1})| + |val(X_{i_2})| + \cdots + |val(X_{i_m})| = |Q|,$$
$$\mu(X_{i_1}) + \mu(X_{i_2}) + \cdots + \mu(X_{i_m}) \leq \tau$$

ここで出てくる解 $\{X_{i_1}, X_{i_2}, \cdots, X_{i_m}\}$ は検索クエリ Q に似た $|Q|$-gram を符号化した最小の変数集合である．変数 $X_i \in V$ が与えられた時，検索アルゴリズムでは以下のように greedy に $|Q|$-gram を発見する．

アルゴリズム まず $|Q|$-gram を $suf(val(X_{l(i)}), |Q| - k), pre(val(X_{r(i)}, k))$ と 2 つの部分文字列に分解して考える．どちらもほぼ同じ操作となるので $suf(val(X_{l(i)}), |Q| - k)$ について考えていく．
$val(X_{l(i)})$ の最右子孫を含み展開長が最大となる

変数を探索していく．$val(X_{l(i)})$ から始めて展開長 L が $|Q|-k$ を超えている場合は右の子へ降りていく．展開長 L が $|Q|-k$ 以下の場合は，その変数を解集合に追加しそこから再び長さ $|Q|-k-L$ の suffix を再帰的に探索していく．新たに変数を追加した時，$\mu(X)$ の合計値が τ を超えた場合は，候補地から除外する．この操作を合計長が $|Q|-k$ になるまで行う．

これとほぼ同様の操作を $pre(val(X_{r(i)}), k)$ に対しても行う．全ての変数に対し上記アルゴリズムを適用することによって，解の候補地をすべて列挙することができる．条件により候補地として出てくる $|Q|$-gram は距離閾値 τ 以下であることが保証できるので，解となるべき候補地を漏らす事は無い．しかし，一方で $F(Q)$ との L_1 距離が τ より大きくなる $|Q|$-gram を候補地として含む場合がある．このような候補地に関しては，後処理として実際に L_1 距離を計算することで除外することが可能である．

定理 5.3 変数 $X \in V$，クエリ Q が与えられた時，候補地発見にかかる時間計算量は，
$O(|T(Q)| \lg\lg(n+\sigma)(\lg|S| + \lg|Q|))$ である．

証明. 本アルゴリズムでは，ESP 木の根から葉に向かって変数を見つけていく．ESP 木の高さは $O(\lg|S|)$ である．またアルゴリズムによって発見される各ノードに対する変数の数は，多くとも $\lg|Q|$ である．
$LeftChild(X), RightChild(X)$ にかかる時間は，
$O(\lg\lg(n+\sigma))$ である．従って，1 つの候補地計算にかかる時間は，$O(\lg\lg(n+\sigma)(\lg|S| + \lg|Q|))$ となる．これが最大で T(Q) 回行われる為，
$O(|T(Q)| \lg\lg(n+\sigma)(\lg|S| + \lg|Q|))$ となる． □

5.4 ポジション計算

候補地の発見と L_1 距離の計算後，条件を満たす $|Q|$-gram が見つかった場合はその全てのポジションの集合 $P(X_i) = \{p \in \{1,2,\cdots,|S|\}; S[p, p+|val(X_i)|-1] = val(X_i)\}$ を計算する．ポジションの計算は変数 X_i から始めて ESP 木の根まで辿っていく．最初に変数 $p = 0$ とする．現在探索している変数 X において (i)X が左の子の場合，p はそのままで X の親ノード X_p に再帰的にアルゴリズムを適用する．(ii)X が右の子の場合，p に $L(X_p) - L(X_i)$ を加えて (i) と同様に再帰する．これを繰り返し根まで到達した時，p には $val(X_i)$ のポジションが入っている為 $P(X_i)$ に p を加える．

定理 5.4 変数 X_i の全ポジション $P(X_i)$ の計算にかかる時間計算量は，$O(occ \lg|S|)$ である．occ は S 上での $val(X_i)$ の出現回数である．

証明. 1 つのポジションを計算する際にかかる時間を考える．ポジションの計算では X_i からアルゴリズムをスタートし根まで辿っていく．この時 ESP 木の高さは $O(\lg|S|)$ であるため，1 つにかかる時間は $O(\lg|S|)$ となる．よって，$val(X_i)$ の全出現を計算するためには $O(occ \lg|S|)$ 必要である． □

6 実験

本章では siEDM を実装し実験した結果について記述する．実験はメモリ 144GB，quad-core Intel Xeon Processor E5540(2.53GHz) のマシンで行った．siEDM を実装した際に用いた rank/select 辞書と GMR は libcds[1] 内のものを使用した．

表 1: 実験に使用したデータセット

	データサイズ	記号数	サイズ (MB)
einstein	467,626,544	139	446
cere	461,286,644	5	440

実験には pizza and chill corpus[2] にある 2 種類のデータを用いた．データの詳細を表 1に示した．

表 2: 実験結果 (ESP 木，特徴ベクトル F，長さベクトル L のサイズ (MB) と構築時間 (sec))

	einstein	**cere**
ESP 木 (MB)	1.18	19.92
特徴ベクトル F(MB)	15.35	227.34
長さベクトル L(MB)	0.59	7.49
構築時間 (sec)	117.65	472.21

表 2は各データの索引構築時間とサイズである．索引サイズの多くを特徴ベクトルが占めていることが分かる．また，サイズもそれぞれ約 16MB と約 256MB と cere の方が einstein に比べ索引が大きい．これはそれぞれの生成規則数が影響していると考えられる．einstein の生成規則数が 305,098 であるのに対し，cere の生成規則数は 4,512,406 である．構築時間に関しては，118 秒と 472 秒であるがこれは siEDM が概ね実時間で動くことを示している．

表 3,4は検索の際にかかった時間等を記載している．CF，DIST，PC がそれぞれ候補地発見・L_1 距離計算・ポジション計算にかかった時間 (sec) で TN と CAND,

[1] https://github.com/fclaude/libcds
[2] http://pizzachill.dcc.uchile.cl/repcorpus.html

表 3: 検索に関する実験結果 (einstein)

データセット		einstein	
クエリ長		50	100
CF(sec)	$\tau = 10$	13.29	15.72
	$\tau = 20$	16.77	13.88
	$\tau = 30$	50.10	16.25
DIST(sec)	$\tau = 10$	0.06	0.02
	$\tau = 20$	0.13	0.06
	$\tau = 30$	0.61	0.20
PC(sec)	$\tau = 10$	0.07	-
	$\tau = 20$	0.05	0.03
	$\tau = 30$	0.01	0.06
♯TN	$\tau = 10$	9,358,327	10,243,531
	$\tau = 20$	11,383,831	8,272,170
	$\tau = 30$	38,746,915	10,470,911
♯CAND	$\tau = 10$	128	0
	$\tau = 20$	1,404	324
	$\tau = 30$	9,098	1,563
♯OCC	$\tau = 10$	434	0
	$\tau = 20$	14,470	6,618
	$\tau = 30$	46,081	34,407

表 4: 検索に関する実験結果 (cere)

データセット		cere	
クエリ長		50	100
CF(sec)	$\tau = 10$	180.81	148.84
	$\tau = 20$	983.46	244.57
	$\tau = 30$	4469.20	874.48
DIST(sec)	$\tau = 10$	0.69	0.28
	$\tau = 20$	29.11	0.40
	$\tau = 30$	2676.59	0.55
PC(sec)	$\tau = 10$	0.000	-
	$\tau = 20$	0.001	0.001
	$\tau = 30$	0.002	0.001
♯TN	$\tau = 10$	97,273,244	64,760,986
	$\tau = 20$	656,441,721	133,129,636
	$\tau = 30$	1,498,193,928	603,891,977
♯CAND	$\tau = 10$	39	14
	$\tau = 20$	23,602	492
	$\tau = 30$	66,738,771	2,548
♯OCC	$\tau = 10$	12	0
	$\tau = 20$	349	37
	$\tau = 30$	964	166

OCC が探索の際に訪問したノード数，候補の $|Q|$-gram 数，発見した数である．τ は移動付き編集距離の閾値である．今回の実験では 10, 20, 30 の 3 通りで実験している．表によると検索時間の多くは CF に費やしている．また，検索時間は閾値が小さいほど高速に計算できている．これは距離閾値が小さい時に枝刈りが特に有効に働いていると考えられる．加えて，表 4の cere の方が einstein より検索時間が低速なのは，生成規則数が cere の方が多いためと考えられる．以上から，圧縮率が高いほど siEDM は高速に検索できる手法である．

7 おわりに

本稿では，ESP の性質を用い，移動付き編集距離 k 以下の部分文字列を検索可能な索引構築アルゴリズムを提案した．実験の結果から実世界の繰り返しの多いテキスト集合にも実行可能なことが示された．実行速度が既存の索引検索と比較して低速であることが分かったので，検索アルゴリズムの高速化が今後の課題である．

参考文献

[1] F. Claude and G. Navarro: Self-indexed grammar-based compression. *Fundamental Informaticae,* 111(3):313-337, 2011.

[2] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich and S. J. Puglisi: LZ77-based self-indexing with faster pattern matching. In *11th Latin American Theoretical Informatics Symposium,* pages 731-742, 2014.

[3] G. Cormode, S. Muthukrishnan: The string edit distance matching problem with moves, *ACM Transactions on Algorithms*,Vol. 3 (1) (2007)

[4] Y. Takabatake, Y. Tabei, and H. Sakamoto: Improved ESP-index: a practical self-index for highly repetitive texts, *In SEA*,pp. 338-350 Copenhagen (DENMARK)(2014)

[5] D. Shapira and J. A. Storer: Edit distance with move operations, *In Proceeding of the 13th Symposium on Combinatorial Pattern Matching.* Lecture Notes in Computer Science, vol. 2373. 85-98.

[6] A. Golynski, J. I. Munro, and S. S. Rao: Rank/select operations on large alphabets:a tool for text indexing. *In Proceedings of the 17th Annual ACM-SIAM Symposiumon Discrete Algorithms*, pages 368-373, 2006.