RD-001

類似文字列検索における LCP 配列を用いた可変長 N-gram 抽出手法の効率化

An Efficent Variable Length N-gram Extraction Using LCP Array for Approximate String Matching

木村 光樹 † 高須 淳宏 ‡ 安達 淳 ‡
Mitsuki Kimura Atsuhiro Takasu Jun Adachi

1 はじめに

類似文字列検索はレコード同定 (record linkage) や web 検索等におけるクエリ修正，単純なスペルチェックや，音声認識や OCR に代表されるような画像認識を行う場合，認識機の性能を補完させる目的で用いられている．このように類似文字列検索は様々なアプリケーションの基礎的な技術として多岐に渡る分野に適用ができるため，その果たす役割は大きい [2, 4, 5, 15, 20]．また近年の情報量の増加 [9] とともにその高速さと正確性がより求められるようになってきた．類似文字列検索で扱う問題は次のように定義される．

ある 2 つの文字列, s, t, 間の類似度を $sim(s,t)$, 距離を $d(s,t)$, 探索対象のデータセットを S, クエリ文字列を Q，k をしきい値だとすると，類似文字列検索では以下の集合を求める問題である．

$$\{s \in S | sim(Q,d) \geq k (\text{or} \quad d(Q,s) \leq k)\} \qquad (1)$$

文字列間の距離や類似度やを表す尺度には, 編集距離, Jaccard Similarity, Cosine Similarity, Dice Similarity など様々に存在する．その中でも編集距離 [11] は類似文字列検索を行うときによく用いられる類似度であり，この類似度に特化したアルゴリズムが数多く研究されてきた．編集距離は，文字列をある文字列に変更するのに必要な編集操作 (挿入・削除・置換) の最低数と定義される．例えば，文字列” 千代田区一ツ橋” と文字列” 千代田区一橋” では前者の文字列の 6 番目の文字を削除すると後者の文字列と一致するので，この場合の編集距離は 1 である．しかしながら，これらの類似度や距離を計算するのには時間がかかることが知られている．一般に，ある 2 つの文字列の文字列長が n, m であるとすると，編集距離の計算には $O(nm)$ の計算量が必要である．検索対象となるデータセットが巨大になれば，距離の計算量も膨大になってしまい，効率的に解候補を削減する必要がある．そのため，近年では類似した文字列同士は，共有する部分文字列が多いということに着目し，N-gram によるの索引付けを利用した高速化手法が数多く研究されている [3, 8]．ここでの N-gram とは文字列中の長さが N である部分文字列の集合のことである．例えば，文字列” 千代田区一ツ橋” の 2-gram は { 千代，代田，田区，区一，一ツ，ツ橋 } となる．N-gram によるの索引付けを用いた類似文字列検索 (図 1 参照) では，この gram を索引語とする転置索引を構築する．表 1 の文字列を 2-gram で索引付けしたときの例を表 2 に示す．問い合わせのときには，クエリ中の gram を求め，その gram を索引語とする文字列リストを転置索引から得る．そして，得た文字列リストをマージすることで解候補を得て，その解候補とクエリとの実際の類似度や距離を求め，最終的な解を求める．この文字列リストをマージするときに，用いる類似度や距離とそのしきい値，クエリから決まる共有する gram の下限値以上の文字列を探すことで解候補の削減を行う．これは，一般には T-occurrence Problem[18] として知られ，この問題を解くためのアルゴリズムが研究の対象となってきている [12, 21].

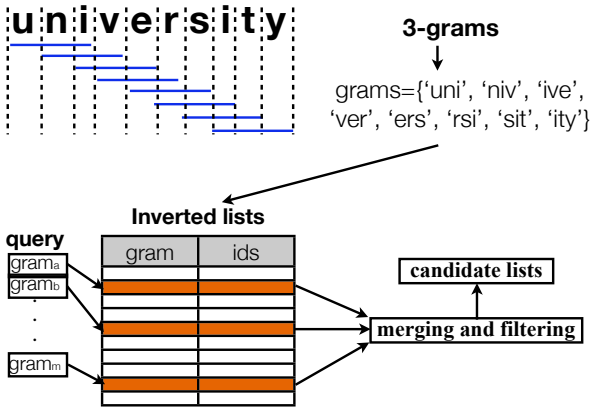

図 1 類似文字列検索のイメージ

しかし，固定長の N-gram では N の値を大きく設定してしまうと，索引がデータ内で使われる文字種の N 乗のオーダーで爆発的に増えていくという問題が指摘されている．また逆に N を小さく設定すると 1 つの文字列 id のリスト内に存在する文字列 id の数が増えてしまうためにマージに時間がかかってしまう．この問題を克服するために gram の長さを可変にして，問題に対応する可変長の N-gram を用いる類似文字列検索手法が提案された [18, 23, 25]．なかでも VGRAM[18] では，テキストマイニングの分野での頻出パターン抽出のようにデータ内に出現するパターンの頻度に着目し，その出現頻度が大きいものを索引語として検索に利用した．しかし，この VGRAM では索引語の抽出に木構造を用いており，時間計算量，空間計算量がともに大きくなってしまうという問題がある．さらに VGRAM の抽出には gram の最長長，最短長，抽出の判断基準となる出現頻度の閾値の 3 つのパラメータを事前に決める必要がある．そのため，パラメータを変更するたびに抽出のための木構造を変更する必要があり，最適パラメータを決定することが困難である．

以上を踏まえて，本研究では類似文字列検索に有効な索引語となる可変長 N-gram を効率よく抽出することを目的とする．我々は，木構造を用いることで必要となる時間計算量，空間計算量を削減するために，線形時間で構築が可能な LCP 配列を用いて可変長 N-gram 抽出する手法を提案する．我々の手法では，データセット中から抽出に必要な接尾辞配列と LCP 配列を一度構築すれば，パラメータを変更してもこの配列を構築し直す必要はない．また，接尾辞配列，LCP 配列，可変長 N-gram の抽出は全て線形時間で行うことができるため，索引語抽出のための時間計算量の大幅な削減に成功した．実データを用いた実験においても，提案する手法の有効性を示すことができた．

† 東京大学大学院

‡ 国立情報学研究所

id	string
0	stich
1	stick
2	such
3	stuck

表 1　string example

gram		string ids
ch	→	0,2
ck	→	1,3
ic	→	0,1
st	→	0,1,2
su	→	2
ti	→	0,1
tu	→	3
uc	→	2,3

表 2　inverted list example

1.1 準備

ここで本稿で扱う表記の定義を行う．

ある文字列 T の長さを $|T|$ で表し，T の i 番目の文字を $T[i](1 \leq i \leq |T|)$ で表す．$T[i]$ から始まり $T[j](i \leq i < j \leq |T|)$ で終わる文字列 T の部分文字列を $T[i;j]$ で表す．文字列 T の文字 $T[i]$ から始まる接尾辞を T_i で表し，2 つの文字列 T, T' 間の接頭辞の最長長を $lcp(T, T')$ で表す．またデータセットを S で表し，その中に出現する文字の全ての集合を Σ で表す．

2 関連研究

2.1 接尾辞配列と LCP 配列

接尾辞配列は文字列中に出現する部分文字列を検索する接尾辞木の空間計算量が大きくなりがちな木構造を，配列構造にしたデータ構造であり，文字列中の部分文字列を高速に検索できる．具体的なデータ構造は，文字列中の全ての接尾辞を辞書順に並べ，各接尾辞の開始位置のインデックスを配列に格納する形となっている．文字列 T の接尾辞を辞書順に並べ替えたときに $T_i(1 \leq i \leq |T|)$ が $k(1 \leq k \leq |T|)$ 番目であるとき，文字列 T の接尾辞配列 SA_T を次式で表す．

$$SA_T[k] = i \tag{2}$$

接尾辞配列では，文字列中の接尾辞を辞書順に並べるため，文字列中に出現するある部分文字列パターンは配列上で隣り合って出現する．そのため，その出現頻度は配列上で出現する最初の位置と最後の位置が分かれば求めることができる．つまり，あるパターン $p(|p| = m)$ が文字列中に出現する回数 $freq_p$ は i, j が次式を満たすとき，式 5 のようになる．

$$i = \underset{1 \leq a \leq |T|}{\operatorname{argmin}}(T_a[1;m] = p) \tag{3}$$

$$j = \underset{1 \leq a \leq |T|}{\operatorname{argmax}}(T_a[1;m] = p) \tag{4}$$

$$freq_p = j - i + 1 \tag{5}$$

これに対し，LCP 配列は接尾辞配列上での隣り合う接尾辞の共通する接頭辞長を格納した配列構造をとっており，ある文字列 T の各配列の値は以下の式を満たす．

$$LCP[i] = lcp(T_i, T_{i+1}) \qquad (1 \leq i < |T|) \tag{6}$$

この LCP 配列と接尾辞配列を用いれば，接尾辞木を容易に構築するアルゴリズムが知られている．そして，接尾辞配列は線形時間で構築することができ [6, 10, 16]，また接尾辞配列が求まっていれば LCP 配列も線形時間で構築することができる [7]．文字列 T=”$mississippi$\$” の接尾辞配列と LCP 配列の例を表**??**に示す．ここで終端文字 \$ はデータセット中に出現しない特殊文字で，データセット中のどの文字よりも辞書順が一番小さいとする．この文字列例で，部分文字列’i’ の出現数は $i = 2$,$j = 5$ であるので，$j - i + 1 = 4$ 回出現することが分かる．

表 3　文字列”$mississippi$\$” の接尾辞配列と LCP 配列

Pos	SA	LCP	suffix
1	12	0	\$
2	11	1	i\$
3	8	1	ippi\$
4	5	4	issippi\$
5	2	0	ississippi\$
6	1	0	mississippi\$
7	10	1	pi\$
8	9	0	ppi\$
9	7	2	sippi\$
10	4	1	sissippi\$
11	6	3	ssippi\$
12	3	-	ssissippi\$

また，LCP 配列は次の特徴をもつ．接尾辞配列上での lcp の値は，一方を固定すると単調減少である．

$$lcp(T_{SA_T[i]}, T_{SA_T[i+1]}) \geq lcp(T_{SA_T[i]}, T_{SA_T[i+2]}) \tag{7}$$

このため接尾辞配列上で任意の 2 地点間の lcp の値はその区間中での最小の LCP 値に等しい．

$$lcp(T_{SA_T[i]}, T_{SA_T[j]}) = \min_{i \leq k < j} LCP[k] \tag{8}$$

2.2 VGRAM

固定長の N-gram を用いると N が大きいと索引の量が爆発的に増えるために，索引付けに時間がかかり，また保持する転置索引サイズも大きくなってしまうことが問題である．しかし，長い gram を共有する文字列は少ないため，ある gram を索引としてもつ転置索引中の転置索引は保持する文字列が少ないために容易に id リストのマージが行える．逆に N の値が小さいと，索引の量は少なく抑えることができ，索引付けにはあまり時間を要さない．しかし，その一方で一つの gram を共有する文字列が増えてしまうために，レコード内の文字列 id が多くなってしまい，文字列 id のリストをマージするのに時間がかかってしまうという問題が生じる．よって固定長 N-gram では，用いる N の値により性能が変わってしまうことが指摘されていた [23]．

これを踏まえて，Li らは固定長の gram での N の大きいとき，小さいときのそれぞれの長所をうまく利用する可変長の N-gram を用いた類似文字列検索手法を提案した [13]．固定長の N-gram では N を大きくすると，索引の量が大きくなってしまうため，Li らはデータ中に出現する頻度が多いものを利用することで索引の量を大幅に増やすことなく，長い gram を検索に利用した．

可変長 N-gram 抽出

具体的な構築方法について紹介する．可変長の N-gram の抽出は次の 3 つの手順によって構成される．

1. gram の最長長 N_{max}，最短長 N_{min}，gram の頻度の閾値 τ の 3 つのパラメータを決める．
2. データ内の文字列から N_{max} の長さの gram を木構造に登録する．木の枝は文字を，木のノードは頻度を管理する．
3. 木の根から幅優先で探索し，ノードの管理する頻度が閾値 τ を上回るとき，

SmallFirst　子ノードのうち最も頻度の少ないものを取り除く．
LargeFirst　子ノードのうち最も頻度の大きいものを取り除く．
Random　取り除く子ノードはランダムに選択する．

の 3 つのうちいずれかのポリシーで枝を刈り，残ったノードをルートノードから連結させたものを gram として抽出し，gram 辞書に登録する．このとき，連結したものの終端ノードが N_{min} の長さを超えるときは，ルートノードからその親のノードが深さ N_{min} に達するまで連結したものも gram 辞書に加えていく．

VGRAM では，構築にかかる時間は M をデータセット中に出現する文字数であるとすると最大で $O(N_{max} \cdot |\Sigma| \cdot M)$ の時間がかかる．さらに可変長 N-gram を抽出するための木構造を作り直す必要があるため，パラメータを変更するごとにこの時間計算量が必要となる．

2.3 頻出パターンマイニング

頻出パターンの抽出問題は，データマイニングの基礎的な問題として数多く研究がなされている [1]．相関ルールを探す問題がよく知られているが，部分グラフ探索問題，部分木探索問題などもある [17, 22]．

なかでも，頻出部分文字列マイニングに焦点を当てる．テキスト分類などを行う場合, 頻出部分文字列はそのテキストの特徴を決める上でも重要な役割を果たす．テキスト中に出現する全ての部分文字列を列挙する手法は LCP 配列を用いた Nagao[14] らや Kasai[7] らの手法がある．

また，テキストの特徴を決める上で重要な極大部分文字列という概念がある．極大部分文字列とは，次のように表される．入力文字列 s 中のある部分文字列 p の出現位置の集合を $P(p) = \{q_1, q_2, ..., q_{freq_p}\}$ とし，$P(p)$ と $c \in R$ が与えられた時，$P(p) - c = \{q_1 - c, q_2 - c, ..., q_{freq_p} - c\}$ とする．ある 2 つの部分文字列 p_1, p_2 が $p_1 = \alpha p_2 \beta (\alpha, \beta$ は空文字を含む部分文字列) がであり，$P(p_1) - |\alpha| = P(p_2)$ を満たすとき $p_1 =_P p_2$ であると定義する．このとき，s 中に出現する全ての部分文字列は $=_P$ の関係により，いくつかの部分文字列の集合に分割することができる．この各集合に属する部分文字列のうち，一番長い文字列を極大部分文字列と呼ぶ．$s = mississippi\$$ であるとき，この文字列の 2 回以上出現する極大部分文字列は {issi} となる．

Okanohara らはこの極大部分文字列を拡張接尾辞配列を用いて高速に列挙する手法を提案した [26]．しかし，これらは長さや頻度に柔軟に対応していないという欠点がある．

また，Tsuboi は pivot と呼ばれる分割点を用いた 3 分割法を用いて，再帰的に頻出部分文字列を列挙する手法を提案した [24]．この手法は，頻出パターンマイニングの基礎的なアルゴリズムである *A-priori* アルゴリズム [17] をもとにしている．*A-priori* アルゴリズムは，頻出でないパターンを含む，それより長いパターンは頻出ではないという考えに基づき，探索空間の枝刈りを行う．Tsuboi の手法ではまず，pivot に選ばれた文字から始まる部分文字列，部分文字列の先頭が pivot より小さな部分文字文字列，大きな部分文字列の 3 つの集合に分割する．このとき *A-priori* アルゴリズムを利用して，pivot から始まる部分文字列の個数がしきい値より小さい場合は，その文字から始まる部分文字列に関しては探索を行わない．しきい値より大きい場合は，その文字を頻出語として抽出し，文字を 1 つすすめ，以下探索空間がなくなるまで，再帰的に探索を繰り返す．pivot より小さな部分文字列の集合，大きな部分文字列の集合に関しては，その中で新たに pivot を定め，やはり再帰的に探索を行う．Tsuboi の手法では，頻度に関して対応できるが，長さに関しては対応できていない．

表 5　LCP 配列の例

num	1	2	3	4	5	6	7	8	9	···
LCP	5	4	6	5	5	6	3	4	2	···

3 提案手法

本章では，我々が提案する LCP 配列を用いた可変長 N-gram の抽出手法について説明する．

提案する手法の目的は，可変長 N-gram を索引語に用いる類似文字列で，検索が効率的に行えるような索引語を抽出することである．そのため，Li らの先行研究を踏まえて，索引語とするデータセット中に出現する部分文字列は，その出現頻度が大きいものとする．

Li らの提案した VGRAM では木構造を用いることで, 抽出索引語の抽出を行っていた．しかし，木構造を用いているために，その構築に時間がかかり，またパラメータチューニングのたびに木構造を作り直す必要があるのが問題である．これに対して，我々は LCP 配列を用いることで線形時間で可変長 N-gram を抽出する手法を提案する．さらにこの手法では，一度 LCP 配列を求めれば配列をパラメータを変更したとしても配列を作り直す必要がないため非常に効率的に索引語となる可変長 N-gram を求めることができる．解くべき問題は以下のようになる．

問題設定

文字列の集合からなるデータセットを S としたとき，S 中に出現する全ての部分文字列のうち出現頻度が τ 以上で，かつその長さが n 以上のものを全て抽出する．このとき索引語数を減らすために，条件を満たすものでも，ある可変長可変長 N-gram の接頭辞となっているようなものは抽出しない．例えば，部分文字列 $abcd$ と部分文字列 abc という 2 つの部分文字列のみが抽出の条件を満たしていたとき，後者は前者の接頭辞となっているので，前者のみを抽出する．

3.1 アルゴリズム

抽出は以下のステップで行われる．

Step.1　データセット中の文字列を全てデータセット中に出現しない特殊文字 ('$ $\notin \Sigma$') によって連結し，一つの文字列を作る．
Step.2　1 で作った文字列の接尾辞配列と LCP 配列を求める．
Step.3　2 で求めた LCP 配列に対して **Algorithm1** を適用することで，可変長 N-gram の抽出を行う．

Step.2 において LCP 配列中の各配列の値は各接尾辞で一番最初に出現する特殊文字より前の部分までの値とする．例えば，文字列”abcde” と文字列”abcdf” の例では lcp の値は 2 となる．

アルゴリズム中で用いる各変数等の説明は表 4 の通りである．

Algorithm の詳しい解説をする．

まず，同じ LCP 値を持つ配列位置の値を同じ bucket に入れる．そして，bucket 内の配列位置の数が τ 個になったら，最小の LCP 値 $(= m)$ が格納されている bucket の LCP 値を抽出候補 (*candidate*) とする (Algorithm 21-22 行目)．このとき，各 bucket 内の数値を最小の LCP 値が格納されている bucket 内に格納されている最も大きい配列位置 $(= s)$ より小さい配列位置は全て破棄する (Algorithm 23-24 行目)．なぜならば，区間

表 4　アルゴリズム中の各変数

Expression or Function	Description
candidate	the output candidate that contains a position and an LCP value
P_k	the bucket of positions with an LCP value of k
$len(P_k)$	return of the number of positions in P_k
P.clean	removal of all buckets
candidate.initialled	the setting of both candidate.position and candidate.lcp to 0

Algorithm 1 Input: LCP, n, τ and N

```
1: i:=1.
2: e:=τ.
3: tmp_n := n.
4: candidate.initialled.
5: while e < N do
6:   while (Σ_k len(P_k) < τ and e < N) do
7:     if LCP[i] < tmp_n then
8:       P.clean.
9:       tmp_n := LCP[i].
10:      i++.
11:      e := i + τ.
12:      if candidate.lcp > n then
13:        Output:candidate.
14:        candidate.initialled.
15:      end if
16:    else
17:      P_{LCP[i]}.push(i).
18:      i++.
19:    end if
20:  end while
21:  m ← the minimum value k of P_k.
22:  s ← the last value in P_m.
23:  for all P_k do
24:    P_k.pop() until the first value in P_k becomes the least
       number which is greater than s.
25:  end for
26:  if candidate.lcp < m then
27:    candidate.lcp := m.
28:    candidate.position := s.
29:    tmp_n := m.
30:  end if
31: end while
32: if candidate.lcp > n then
33:   Output:candidate.
34: end if
```

$[s-\tau, s]$ 内の配列位置の LCP の最小値は m であるので，式 8 により，この区間では長さ m の部分文字列しか頻度が τ 以上で出現していないこととなるためである．抽出は *candidate.lcp* より小さい候補が見つかったときに，抽出を行う (Algorithm 7-15 行目)．これは，抽出する可変長 N-gram のうちある可変長可変長 N-gram の接頭辞となっているようなものは抽出しないようにするためである．もし，次の区間 $[s+1, s+1+\tau]$ で m より大きい値 m' が見つかったとすると，区間 $[s-\tau, s+1+\tau]$ の共通接頭辞長は，やはり式 8 により m となるが，これは区間 $[s+1, s+1+\tau]$ の長さ m' の接頭辞の先頭 m 文字は $[s-\tau, s]$

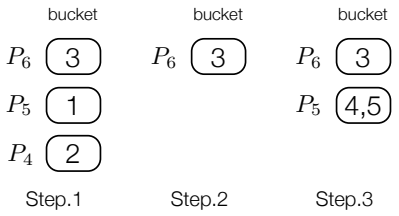

図 2　Algorithm の適用例

ことを意味することとなる．そのため，*candidate.lcp* より大きな値が候補となるときには *candidate* をその値に書き換える (Algorithm 26-29 行目)．逆に $s+1$ 以降の LCP の配列位置で m より小さい値が見つかったとし，このときの配列位置を s' とし，s' 以前で $candidate.lcp = 3$ でかつ，そのときの共通接頭辞を abc とする．また，s' のときの LCP の値を 2 であるとする．このとき，s' 以降も LCP の値が 3 以上であったとしても，共通している接頭辞は配列位置 s' のときの LCP 値が 2 であるために，abc を共有することはない．よって，抽出は *candidate.lcp* より小さい値が見つかったときに抽出を行わなければならない．

この操作を全ての配列を走査するまで繰り返す．以上によりこのアルゴリズムによる抽出に要する時間は M をデータセット中に出現する文字数であるとすると，各配列を一回参照するだけでよいので，$O(M)$ の時間で終わる．またパラメータをチューニングする際，パラメータを変更したとしても，LCP 配列を作り直す必要がなく，この Algorithm を適用するだけでよくパラメータの変更に柔軟に対応できる．

具体的な例を用いてより詳しく **Algrithm** の詳細について述べる．あるデータセットから表 5 のような LCP 配列が得られたとする．長さが 3 以上で出現頻度が 3 以上のもの，つまり $n=3, \tau=3$ のときの抽出過程を紹介する．まず，num=1 から始めて，まず LCP[1]=5 であるので，bucketP_5 を生成し，その中身に num=1 を入れる．同様に bucket の中身の総数が $\tau=3$ に等しくなるまで bucket を追加していくと図 2 中の Step.1 のような bucket が得られる．このとき，P_4 の bucket が最小であるので *candidate.lcp* を 4 にセットし，*candidate.position* を P_4 の bucket 内の最後尾の値である 2 にセットする．そして，他の bucket 内で 2 より小さい配列位置を削除する．すると図 2 中の Step.2 のようになる．次にまた bucket の中身の総数が τ に等しくなるまで bucket に配列位置を追加していくと図 2 中の Step.3 のようになる．このとき，*candidate.lcp* は 5 になるので，これは前回の *candidate.lcp* より大きいため *candidate.lcp* を 5 に *candidate.position* を 4 に書き換える．そして，また bucket に配列位置を追加していくことになるが，LCP[7]=3 であり，これは *candidate.lcp* が保持している値，5 より小さいため *candidate.lcp* と *candidate.position* を求めるものとして抽出し，現段階で保持している bucket を全て破棄する．これらの

表 6　実験で使用したデータセット

Data	Size	Averaged length	$\|\Sigma\|$	Description
EnglishDictionary	635K	9.4	26	english dictionary supplied in SAISAP[19]
DBLP	78M	104.5	93	bibliography data supplied in DBLP??

表 7　Suffix array と LCP array 構築時間

Dataset	Suffix Array(sec)	LCP array(sec)
English Dictionary	0.13	0.08
DBLP	70.32	24.92

操作を LCP 配列を全て走査するまで繰り返す．

4　評価実験

4.1　実験環境とデータセット

本研究では，提案手法と Tsuboi の提案したアルゴリズムの比較を行った．Tsuboi の提案したアルゴリズムと本手法が想定する問題は異なるため，提案するアルゴリズムで抽出する頻出語の長さのしきい値 n を $n = 1$ とした．実験環境は，OS が Debian GNU/Linux 6.0.5 であり，CPU が Intel Xeon X5492 3.4GHz，メインメモリが 64GB である計算機を使用した．実装言語は C である．

本研究で用いたデータセットの詳細は表 6 に載せる．English Dictionary を用いた時には頻度のしきい値，τ を 50 から 250 まで 50 刻みで変化をさせた．また，DBLP のデータセットを用いた時は τ を 500 から 2000 まで 500 刻みで変化をさせた．それぞれの場合の提案手法と Tsuboi の手法で抽出にかかる時間の比較を行った．

4.2　結果

提案手法では接尾辞配列と LCP 配列を構築しなければならない．そこで，まずそれぞれの配列の構築にかかる時間を計測した．結果は表 7 に載せる．次に実際に抽出に要する時間の計測を行った．提案手法では，配列は既にできているという想定のもとで提案するアルゴリズムを走らせて時間を計測した．

それぞれの結果を図 3(English Dictionary)，図 4(DBLP) に載せる．グラフの縦軸は，頻出語抽出に要した時間を示しており，横軸は頻度のしきい値 τ を表している．また，各 τ のときに抽出された頻出語の個数を表 8 に示す．図，表ともに LCP が提案手法であり，mfs が Tsuboi の手法である．どちらのデータセットの場合でも提案手法が，3 倍-4 倍の性能向上が見られ，提案手法の有効性が示されたと言える．また抽出される頻出語数も特に DBLP の場合であれば，提案手法は Tsuboi の手法と比較して約 1/2 となっており，抽出される頻出語がよく圧縮されていることが分かる．

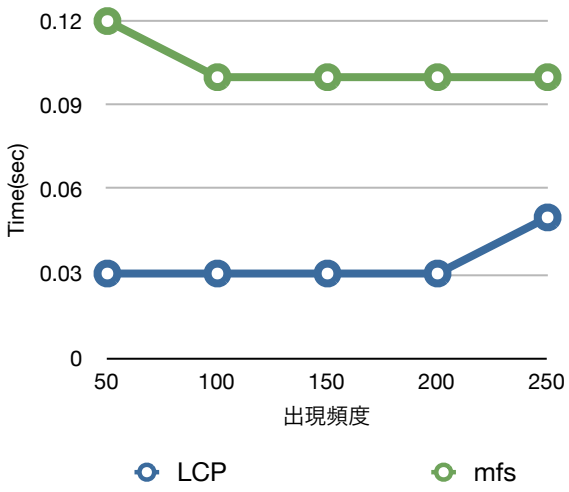

図 3　索引語抽出時間比較 (English Dictionary)

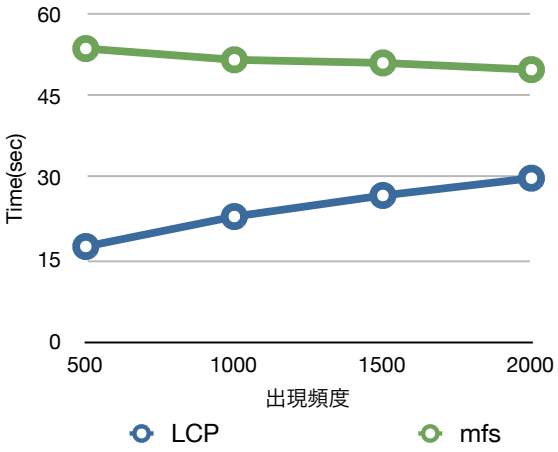

図 4　索引語抽出時間比較 (DBLP)

5　まとめ

類似した文字列を検索するには，近年その共通部分の大きさに着目し gram による索引付けを用いる研究が盛んに行われている．しかしながら，gram の長さによりその検索システムには大きな性能の差が生じることが指摘されてきた．そのため，用いるデータによって gram の長さを求める必要がある．これまでの研究では，その実現を木構造を用いていたため，索引構築時の時間計算量，空間計算量が大きく，またパラメータのチューニングのたびに木構造を作り直す必要があり，非常にコストが大きかった．これに対して，我々は LCP 配列を用いて線形時間で索引語を抽出する手法を提案した．本手法は，一度 LCP 配列をつくっておけばパラメータの変更のときも提案したアルゴリズムを適用させるだけでよいというのも特徴の一つである．また，実データを用いた評価実験においても 3-4 倍程度の速度向上が見られた．今後の課題として，クエリ中の索引語を効率的に抽出できるようにすることが挙げられる．

参考文献

[1] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. pp. 3–14, 1995.

[2] Akiko AIZAWA, Atsuhiro TAKASU, Keizo OYAMA, and Jun ADACHI. Record linkage of multi-source databases: Research trends. *NII journal*, Vol. 8, pp.

表 8　頻出語数

frequency(τ)	50	100	150	200	250
English(LCP)	3198	1718	1147	888	704
English(mfs)	4164	2091	1378	1031	803
frequency(τ)	500	1000	1500	2000	−
DBLP(LCP)	49817	25408	17148	12968	−
DBLP(mfs)	105059	50378	33186	24687	−

43–51, 2004.

[3] Alexander Behm, Shengyue Ji, Chen Li, and Jiaheng Lu. Space-constrained gram-based indexing for efficient approximate string search. In *Proceedings of ICDE '09*, pp. 604–615, 2009.

[4] Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. Approximate string joins in a database (almost) for free. In *Proceedings of VLDB '01*.

[5] Marios Hadjieleftheriou and Chen Li. Efficient approximate search on string collections. *Proc. VLDB Endow.*, Vol. 2, No. 2, pp. 1660–1661, 2009.

[6] Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*.

[7] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. Linear-time longest-common-prefix computation in suffix arrays and its applications. pp. 181–192. Springer-Verlag, 2001.

[8] Min-Soo Kim, Kyu young Whang, Jae-Gil Lee, and Min jae Lee. n-gram/2l: A space and time efficient two-level n-gram inverted index structure. In *VLDB*, pp. 325–336, 2005.

[9] Masaru Kitsuregawa and Toyoaki Nishida. Special issue on information explosion. *New Generation Computing*, Vol. 28, No. 3, pp. 207–215, 2010.

[10] Pang Ko and Srinivas Aluru. Space efficient linear time construction of suffix arrays. *Journal of Discrete Algorithms*, Vol. 3, No. 24, pp. 143 – 156, 2005.

[11] Vladimir I Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, Vol. 1, No. 1, pp. 8–17, 1965.

[12] Chen Li, Jiaheng Lu, and Yiming Lu. Efficient merging and filtering algorithms for approximate string searches. *Data Engineering, International Conference on*, Vol. 0, pp. 257–266, 2008.

[13] Chen Li, Bin Wang, and Xiaochun Yang. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *Proceedings of VLDB '07*.

[14] Makoto Nagao and Shinsuke Mori. A new method of n-gram statistics for large number of n and automatic extraction of words and phrases from large text data of japanese. In *Proceedings of COLING '94*, Vol. 1.

[15] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, Vol. 33, pp. 31–88, March 2001.

[16] Ge Nong, Sen Zhang, and Wai Hong Chan. Two efficient algorithms for linear time suffix array construction. *IEEE Transactions on Computers*, Vol. 60, pp. 1471–1484, 2011.

[17] Anand Rajaraman and Jeffrey D Ullman. Mining of massive datasets. *Lecture Notes for Stanford CS345A Web Mining*, Vol. 67, No. 3, p. 328, 2011.

[18] Sunita Sarawagi and Alok Kirpal. Efficient set joins on similarity predicates. In *Proceedinfs of SIGMOD '04*.

[19] International workshop on similarity search and applications(sisap). `http://www.sisap.org`.

[20] Esko Ukkonen. Approximate string matching with q-grams and maximal matches. Technical report, 1991.

[21] Chuan Xiao, Wei Wang, and Xuemin Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *Proc. VLDB Endow.*, Vol. 1, No. 1, pp. 933–944, 2008.

[22] Xifeng Yan and Jiawei Han. gspan: graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pp. 721 – 724, 2002.

[23] Xiaochun Yang, Bin Wang, and Chen Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *Proceedings of SIGMOD '08*.

[24] TSUBOl Yuta. Mining frequent substrings(natural language understanding and models of communication). *IEICE technical report. Natural language understanding and models of communication*, Vol. 103, No. 408, pp. 79–86, 2003-10-31.

[25] 木村光樹, 高須淳宏, 安達淳. 頻度情報を用いた類似文字列検索のための可変長 n-gram. 情報処理学会研究報告. 情報学基礎研究会報告, Vol. 2011, No. 13, pp. 1–8, 2011-07-26.

[26] 岡野原大輔, 辻井潤一. 全ての部分文字列を考慮した文書分類. 情報処理学会研究報告. 自然言語処理研究会報告, Vol. 2008, No. 90, pp. 59–64, 2008-09-17.