2P-8

類似文字列検索のためのSuffix Treeを用いた可変長N-gram

木村　光樹 † 高須　淳宏 ††　安達　淳 ††
† 東京大学大学院　†† 国立情報学研究所

1　はじめに

近年，表記の揺れを包含した情報の検索，すなわち類似文字列検索は表記の揺れを考慮してデータベースを参照したりノイズを含むデータベースをクリーニングする際に必須の技術となっている．

類似文字列検索では固定長のN-gramを用いた手法が使われることが多い．N-gramとは，編集距離で表される類似度の近い文字列を検索するときに用いられ，文字列を固定サイズの部分文字列(gram)に分割して扱う手法である．これに対して，近年ではVGRAM[1]など可変長のN-gramを用いて探索効率を向上させた手法が研究されている．しかし，VGRAMでは，部分文字列の長さを自動的に決めるアルゴリズムは提案されているが，パラメータを事前に3つ定めねばならず，パラメータチューニングのコストが高い．そこで本稿ではSuffix Treeを用いたgram長に関するパラメータを必要としない可変長N-gramを提案する．我々の提案する可変長N-gramは構築が容易なだけでなく，類似文字列の探索も効率良く処理できる．可変長N-gramの先行研究であるVGRAMと比較した評価実験では，探索コストはVGRAMとほぼ同様の結果となった．

2　提案手法

本論文では，Suffix Treeを用いた可変長N-gramを提案する．まず，可変長N-gramの選択について述べる．次にそれを用いた類似文字列検索方法について述べる．

2.1　gram抽出

我々の提案する可変長N-gramは，先行研究のVGRAMと同様に頻度情報を使ってgramの長さを最適化する．しかし，VGRAMではgramの最短長(q_{min})，最長長(q_{max}),gramの頻度閾値(T)の3つのパラメータが必要である．それに対して提案手法ではgramの頻度の閾値Tの1つのみを必要とする．我々の提案するgram抽出は以下の3つの手順で構成される．

(1) 辞書構築用のデータ内の各文字列をSuffix Treeに登録する．このとき，各ノードが何回出現しているかを数えておく．
(2) 木の深さが3以上のノードに対して，その頻度が閾値以下のノードを枝刈りする．
(3) 残ったノードをgramとして抽出する．

このようにして抽出されたgramは木構造のままで保持しておく．このgramの辞書を$D(S)$とする．

2.2　gramへの分割

gram抽出によって得られた辞書を用いて文字列をgramへと分割する手法について述べる．各文字列について最長一致するgramを$D(S)$の中から探し，gramの出現位置とともに蓄積する．我々はVGRAMで提案されていたVGENという関数を適用した．ここで引数として必要なq_{min}には，gramの最小単位の2を代入する．VGENの擬似コードをAlgorithm 1に示す．

このとき$s[i]$は文字列sのi番目の文字を表し，$s[i;j]$は文字列sの$s[i]$から始まり，$s[j]$で終わる部分文字列を表す．

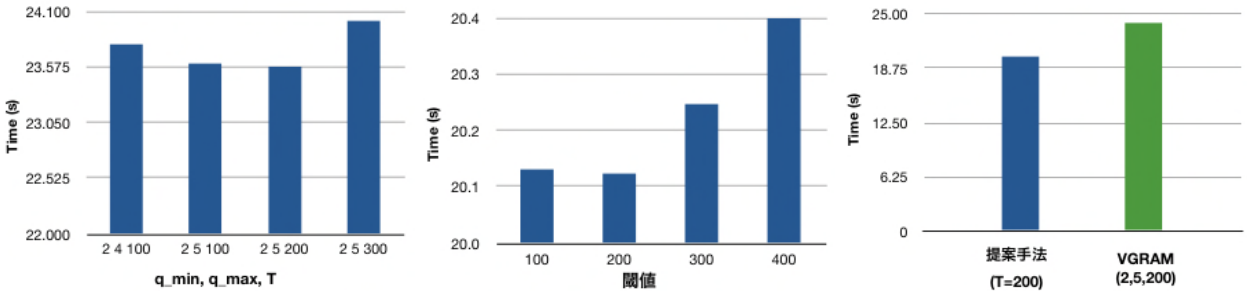
Algorithm 1 VGEN

Input: gram-dictionary D,string s,bounds N_{min}
Output: a set of positional gram,VG
position $i=0$; $VG=\{\}$;
WHILE($i \le |s| - q_{min}$){
  Find a longest gram in D using the trie to match a substring of t of s starting at position i;
  IF (t is not found) $t = s[i; i+q_{min}-1]$;
  IF (positional gram (i,t) is not subsumed by any positional gram in VG)
    Insert (i,t) to VG;
  $i = i+1$
}
RETURN VG;

2.3　NAG

次に各文字列のNAG(the number of affected grams)を計算する．NAGは，文字列sにk回の編集操作を行ったときに，最大でいくつのgramが影響を受けるかを表したものである．NAGを用いると，2つの文字列間で共有するgram数の下限値は次のよう

Variable Length N-gram Using Suffix Tree for Approximate String Matching
† Mitsuki Kimura(mick@nii.ac.jp)
†† Atsuhiro Takasu(takasu@nii.ac.jp)
†† Jun Adachi(adachi@nii.ac.jp)
The University of Tokyo (†)
National Institute of Informatics (††)
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo,101-8430 Japan



(a) VGRAM　(b) 提案手法　(c) 比較

図 1: パラメータとクエリ処理時間

に表すことができる．

$$B_vc(s_1, s_2, k) = max\{|VG(s_1)| - NAG(s_1, k), |VG(s_2)| - NAG(s_2, k)\} \quad (1)$$

文字列 s の i 番目の文字 $s[i]$ に編集操作を行うとき，$s[i]$ を含む gram はこの編集操作によって必ず影響を受ける．また，$s[i]$ を含まない gram も影響を受ける gram が存在する．このような gram を $D(S)$ の gram 木と $D(S)$ 内の gram を反転させ登録した gram 木，$D(S)$ 内の最長 gram の gram 長を用いて求める [1]．s 中の各文字に対してその文字に編集操作を行ったときに影響を受ける可能性のある gram 数を求め，その最大値を文字列 s の NAG とする．

2.4 類似文字列検索手法

類似文字列検索には MergeCount[2] と呼ばれる逆引きのリストを用いた手法を利用する．クエリを Q としたとき，MergeCount では以下の 3 つのフィルタリングが用いられる．

Length filtering $|s| > |Q|$ となる s を除去．
Position filtering 共通する gram を持っているが，その出現する位置がクエリと文字列で k 以上離れているものを除去．
Count filtering クエリと文字列で，共有する gram 数が共有する下限値より少ないものは除去．

提案手法に適用するために，Position Filtering の gram 分割では gram 辞書 $D(S)$ を使い，関数 VGEN を用いる．また，Countfiltering の共有する gram の下限値には式 (1) で求まるものを用いる．

3 評価実験

実験ではエントリー数が 69,069 の英単語辞書を用いて実験を行った．単語長は平均 9.4 文字であった．全ての単語を用いて，Suffix Tree を用いた可変長 N-gram，VGRAM を生成した．また，全単語中から任意に選んだ 1,000 個の単語を編集距離 1 の文字列へと変換し，検索用のクエリとした．類似文字列の検索には，MergeCount を用いてフィルタリングを行い，提案手法，VGRAM のそれぞれで最適なパラメータを求めた．

図 1(a) に VGRAM を用いて各パラメータを変化させたときの，クエリ処理に要する時間の変化を示し，図 1(b) に提案手法で閾値を変化させたときの処理時間の変化を示す．それぞれの最適なパラメータのときのクエリ処理時間を比較すると，提案手法が優位であることが分かる (図 1(c))．しかし，VGRAM と比較すると，提案手法では Suffix Tree を用いているため可変長の N-gram を生成するときに時間がかかり，メモリの使用量も大きくなってしまう．この二つを減らすことが今後の課題である．

4 おわりに

編集距離で類似度が表現される類似文字列検索において，Suffix Tree を用いた可変長 N-gram の生成方法を提案した．提案手法ではパラメータチューニングのコストを削減することができ，提案手法を用いた類似文字列検索では，既存の手法と同等の性能を示すことができた．

参考文献

[1] Li et al. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, 2007.

[2] Sarawagi et al. Efficient set joins on similarity predicates. In *SIGMOD*, 2004.