Vol.2011-DBS-152 No.13
Vol.2011-IFAT-103 No.13
2011/8/3

頻度情報を用いた類似文字列検索のための可変長 N-gram

木村光樹[†1]　高須淳宏[†2]　安達淳[†2]

類似文字列検索では類似した文字列同士は共有する部分文字列が多いことに着目し，gram と呼ばれる長さ N の部分文字列に文字列を分割し，それを索引語とする転置索引を構築し，検索に利用する gram ベースの索引付けを用いる手法が数多く研究されている．しかし，N を大きくすると索引数が N 乗のオーダーで増えていくため，N の値を大きく設定すると索引付けや問い合わせに時間がかかってしまうので N の値は小さい値に設定されることが多い．その一方で，N の値が大きいと長い部分文字列を共有する文字列は一般的に少なくなるため解候補の削減が容易に行えるという利点がある．このことを踏まえて，データ中に出現する長い gram のみを利用する，N の値を可変長に拡張した VGRAM という手法が提案された．しかし，VGRAM では可変長の gram の抽出をするために gram 長の上限値，下限値，抽出の判断基準となる頻度の閾値の 3 つのパラメータを事前に決めなければならない．これらのチューニングコストを考えると，VGRAM は実用的ではない．そこで我々は，Suffix Tree を利用して gram の長さに関するパラメータを必要としない新しい可変長 N-gram を提案する．我々の手法では，tree 上での出現頻度を考慮して文字列の最適な長さの gram の抽出を行う．評価実験では，VGRAM と比較してパラメータチューニングのコストが少なく，高速に類似文字列検索できる端緒を示すことができた．

Variable Lengh N-gram Using Gram Frequency for Approximate String Matching

Mitsuki Kimura,[†1] Atsuhiro Takasu[†2]
and Jun Adachi[†2]

Fixed-length grams(N-gram) are widely used for the problem of approximate string matching, for similar strings share many grams in common. Recently, VGRAM improves the efficiency of the matching by optimizing the size of each gram. Although VGRAM sets the length of grams, it requires three parameters: the upper bound and the lower bound of the length of the grams, and the frequency criterion of the gram partitioning. The cost of the parameter tuning is heavy. In this paper, we propose a new variable length N-gram, which requires only one parameter and realizes efficient approximate substring matching. Our variable length N-gram automatically optimize the size of grams by using Suffix Tree. Experimental results compared with VGRAM indicate that our method is as efficient in finding similar substrings as VGRAM.

1. はじめに

類似文字列検索は大量のオブジェクト中からクエリと似たオブジェクトを探す Similarity Search の対象オブジェクトとして文字列を扱うものである[19]．その上，人が扱う情報は文字列として扱えることが多いため，その応用範囲は広い[5]．

類似文字検索は，単純な spell check や web 検索でのクエリ修正，データベース中のデータクリーニングや record linkage にも用いられる[3]．record linkage とは単一あるいは複数の情報源にまたがるレコードの集合から，同一の実態を参照するペアまたはグループを識別する問題である[1]．

例えば，’千代田区一ツ橋 2-1-2’ と’千代田区一橋 2-1-2’ という 2 つの文字列を考える．人が見たとき，後者は前者の間違いだと理解されることが多いが，計算機で判断する場合は，完全に一致していないものは別のものとして扱われる．このため，探したいものを見つけられなかったり，データベースを作成するときに不要なレコードをつくってしまう問題がある．

類似文字列検索が適用できる範囲は広く，様々な技術の基幹として用いることができるが，情報量の増加とともに[7][18] 高速さと正確性がより一層求められるようになってきた[13]．

文字列間の類似度を表す尺度には，編集距離，Jaccard Similarity, Cosine Similarity, Dice Similarity など様々に存在する[12]．その中でも編集距離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[14]．しかしながら，これらの類似度を計算するのには時間がかかることが知られているため，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 ベース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[2][4][6][9][15][17]．すなわち，クエリ中の各 gram を索引語とする転置索引中のレコード内の文字列 id のリストの中に頻出する文字

†1 東京大学大学院情報利工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
†2 国立情報学研究所
National Institute of Informatics

Vol.2011-DBS-152 No.13
Vol.2011-IFAT-103 No.13
2011/8/3

列 id 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 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．

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．また 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．なかでも VGRAM[10] 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[11)21)] 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 を求めるためには 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り，チューニングコストが大きい．そこで我々は，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．我々の手法ではチューニングコストを小さくすることで可変長 N-gram の構築を容易にし，VGRAM と同程度の検索スピードを実現した．

2. 関 連 研 究

2.1 類似文字列検索

編集距離[8)] は類似文字列検索でよく用いられる類似度である．文字列間の編集距離は，ある文字列を別の文字列へと変換するのために行う編集操作の最小回数で表される．編集操作とは，挿入・削除・置換の 3 つの操作である．それぞれの操作を行うごとに距離が 1 ずつ増えていく．

例えば s_1 ='misisippi'，s_2 ='misisippii' であるとき，文字列 s_1 を文字列 s_2 に変換するためには $s_1[5]$ ='s' を削除し文字列の最後に文字'i' を挿入すれば文字列 s_1 は文字列 s_2 へと変換できる．よって文字列 s_1 と文字列 s_2 間の編集距離は 2 である．この文字列 s_1，s_2 間の編集距離を $ed(s_1, s_2)$ と表すこととする．文字列データの集合を S としたとき，クエリ Q と類似した文字列を検索するとは，ある閾値 k を用いて，次式を満たす $s \in S$ となる文字列 s を全て見つけることである．

$$ed(s, Q) \leq k \quad (1)$$

しかしながら，文字列 s_1 と s_2 の編集距離を計算するためには一般に $O(|s_1||s_2|)$ の計算量がかかり，類似度の計算には時間がかかってしまうことが知られている．

表 1 文字列データ
Table 1 Strings

id	string
0	stick
1	stich
2	such
3	stuck

表 2 転置索引
Table 2 Inverted lists

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

2.2 gram ベースの索引付けと T-occurrence Problem

文字列を s とし，その長さを |s| で表す．$|s| = m$ であるときに，文字列 s の i($1 \leq i \leq m$) 番目の文字を $s[i]$ で表す．また $s[i]$ から始まり，$s[j](1 \leq i < j \leq m)$ で終わる文字列 s の部分文字列を $s[i; j]$ で表す．s='mississippi' であるとき，$|s| = 11$,$s[5]$ ='i',$s[2; 5]$ ='issi' となる．

N を自然数とする．N-gram とはある文字列 s の長さが N の部分文字列の集合である[22)]．文字列 s の N-gram の集合 $G(s, N)$ は，次のように定義できる．

$$G(s, N) = s[x; x + N - 1] \quad (1 \leq x \leq m - N + 1) \quad (2)$$

gram は転置索引を構築するのに用いられ，このとき各索引語は gram の一つひとつが割り当てられることとなり，各レコード内にその gram を含む文字列 id を挿入していくことで，転置索引を構成する．例えば，表 1 中の文字列を 2-gram で索引付けを行った場合，転置索引は表 2 のようになる．

さて，前述の文字列 s_1 ='misisippi' と文字列 s_2 ='misisippii' において $G(s_1, 3)$ ={'mis', 'isi', 'sis', 'iss', 'ssi', 'sip', 'ipp', 'ppi'}，$G(s_2, 3)$ ={'mis', 'isi', 'sis', 'isi', 'sip', 'ipp', 'ppi', 'pii'} となる．'mis', 'isi', 'sis', 'sip', 'ipp', 'ppi' といった gram を共有していることが分かる．このように，類似した文字列同士は gram を多く共有する．類似度の閾値を決めれば，2 つの文字列ではその閾値に依存した共有する gram 数の下限値を決めることができる．この値を T とすると，類似文字列検索は次のような問題を解くことで解候補の削減を行うことができる．

T-occurrence Problem $G(Q, N)$ 中の各 gram を索引語とする転置索引中の各レコード内の文字列 id のリストをマージしたときに T 回以上出現する文字列 id を探す．

文字列中のある文字列に編集操作を加えると，その文字を含む gram は影響を受ける．長

Vol.2011-DBS-152 No.13
Vol.2011-IFAT-103 No.13
2011/8/3

さ N の gram を索引語としたとき，1 つの文字あたり最大で N 個の gram に含まれることになる．また一つの文字列からは $|s|-N+1$ 個の索引語を得ることができる．よって編集距離を類似度に用いて編集距離が k 以内の文字列を検索したいとき，この T の値は次のように見積もることができる．

$$T = \max(|Q|,|s|) - N + 1 - N \cdot k \quad (3)$$

Algorithm 1 MergeOpt(r, T, I)

Let $A = l_1, l_2, ..., l_t$ be the record lists of index I in decreasing order of length corresponding to the t words $w_1, ...w_t$ of r
Compute cumulative$W_t(l_i) = \sum_{j=1}^{i} weight(w_j)$
$L = l_1, l_2, ..., l_k$ such that k is the largest index for which cumulative$W_t(l_k) < T$
Inserts frontiers of lists $S = A - L$ in a heap H.
while H not empty **do**
  pop from H current minimum record
  m along with total weight $m.w$ of all
  lists in H where m appears
  push in H next records from lists in S
  that popped.
  for $i = k$ down to 1 **do**
    if($m.w$+ cumulative$W_t(l_i) < T$)
    exit-for;
    search for m in l_i using a doubling
    binary search method, and if
    found,
      increment $m.w$ with weight(w_{l_i}).

2.3 マージ手法

クエリ中の各 gram を索引語とする転置索引中のレコード内の文字列 id の各リストをマージし，T-occurrence Problem を解くための MergeOpt[15] という手法について説明する．MergeOpt は Sarawagi らが類似文字列検索よりも広義で使われる，Similarity Join の手法の一つとして提案した手法である．ある gram g を索引語とする転置索引中の文字列 id のリストを l_g とおいたとき，l_g をその id を含む長さでソートし，ヒープ構造を用いて文字列 id へのアクセスを少なくすることで解候補の抽出を高速化した．

具体的なアルゴリズムを Algorithm1 に載せる．MergeOpt ではまず，クエリを gram に分割したときのそれぞれ gram の l_g を id を多く含む順にソートする．そして，各 id がもつ重み (cumulativeW_t) を足し合わせたときに，T を超えない個数のうち最も大きい値を k とする．ソート済みのリスト l_g から大きい順に k 個とり，そのリストを L とする．残りのリスト ($S = A - L$) 個リストの先頭の id をヒープ (H) の中に入れる．そして，H 内の最小の id を m とし,H 内の m を全て取り出し，その重みを計算する．そして L 内の保持している id が少ない順に一つひとつ binary search をしていき，m が見つかる度に重みを再計算していく．そして，もしその重みが T を超えたときには，結果の id リスト I にその id,m を追加する．m の重みが T を超えるか，L 内の最長リストまで探索し終えたときに，S 内の m であった id を持っていたリストの m の次の id を H に追加する．この操作をヒープ H が空になるまで続ける．

これを gram ベースの索引付けを用いた類似文字列検索に用いるためには，id のもつ重みを 1 とし，$k = T - 1$ として計算すれば，そのまま T-occurrence Problem を解くことができる．

2.4 VGRAM[10]

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．

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．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．

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，

Vol.2011-DBS-152 No.13
Vol.2011-IFAT-103 No.13
2011/8/3

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．

具体的な構築方法について紹介する．表 1 の文字列から VGRAM を構築する．このとき，$N_{min}=2$,$N_{max}=4$,$\tau=2$ を用い，枝刈りのポリシーには Random ポリシーを用いるものとする．

まず文字列'stick' を木に登録する．木に登録するときは，$N_{max}=4$ であるのでこの長さの gram を登録する．つまり，'stic' をまず木に登録し，N_{min} 以上の深さの葉ノードの出現頻度を 1 ずつ増やす．そして，次に'tick' を登録し，各葉ノードの出現頻度を更新する．ここで長さ N_{max} での文字列の登録は終わるが，このとき抽出する gram の長さは N_{min} 以上 N_{max} であるので，登録できていない N_{min} 以上 N_{max} 未満の長さの gram を考慮し，gram'ick' と'ck' を同様に木に登録する．これを全てのデータ中の文字列に対して行う．

この完成した木を頻度情報をもとに枝刈りすることで gram を抽出する．ここでは $\tau=2$ であるので深さの浅い方から見ていき，深さが N_{min} となるノードでその頻度情報を見る．頻度が τ 以下であればそのノードより深いところにあるノードは全て枝刈りを行う．もし τ を超えるノードが見つかれば，その子ノードからポリシーに従い子を一つ選び，それ以外のノードは全て枝刈りをする．そして残ったノードに対して，再帰的に枝刈りの手法を用いていき，終端のノードに辿り着くまで枝刈りを行っていく．これを行うと，結果として gram 辞書 D は D={'ch','ck','ic','sti','su','tu','uc'} が抽出される．

この可変長 N-gram を用いて索引付けを行うとき，VGRAM では gram の長さが様々であるために，gram 辞書内にある全ての gram を用いて 1 つの文字列を索引付けしようとすると固定長の N-gram を用いたときに比べ索引数が多くなってしまう可能性がある．このことを踏まえて，索引付けを行う場合には次のポリシーに従って，できるだけ簡素な索引付けをする必要がある．

(1) gram 辞書内の最長一致する gram を索引語とする．

(2) 他の gram の部分文字列となるような gram では索引語としない．

(3) gram 辞書内に存在 gram が存在しないときは，長さ N_{min} の gram を索引語とする．

実際の例について説明する．gram 辞書を用いてある文字列 s を gram に分割したとき，その gram の文字列中での開始位置と gram のペアの集合を $VG(s)$ とする．

gram 辞書，D={'uni','iv','ivr','ver', 'vers', 'rs','sit', 'ty', 'ity'} を用いて,$N_{min}=2$,$N_{max}=4$ とし，s='university' の索引の集合 VG(s) は次のようにして求められる．まず，先頭の文字 s[1]='u' から始まる gram 辞書中の gram で文字列 s の部分文字列となっているものは，'uni' が見つかる．よって，その文字列中での開始位置とともに (1,'uni') を記憶する．次に，文字を一つ進め，s[2]='n' から始まる gram を辞書中から同様に探す．

しかし，これは先に見つかっている gram(1,'uni') の部分文字列となっているので，この gram は索引語としない．同様に，索引付けのポリシーに従って，文字を進めながら gram で分割していくと結果として索引語の集合は $VG(s)=\{(1,uni),(3,iv),(4,vers),(7,sit),(8,ity)\}$ が得られる．

索引付けと同様に，gram 長が固定でないためにクエリと文字列が共有する gram 数は簡単には求まらない[20]．例えば，索引付けの例と同様の，gram 辞書 D と文字列 s='university' のときを考える．このとき $VG(s)=\{(1,uni),(3,iv),(4,vers),(7,sit),(8,ity)\}$ となる．この $s[4]$ ='e' を削除することを考える．このとき，'e' を含む gram，(2, vers) が，この編集操作で影響を受けることは自明である．しかし，(2, iv) という gram は $s[4]$ ='e' が消去されたとき，D 内に gram，'ivr' が含まれているため，(2,ivr) と変化する．この例のように直接編集操作とは関連がないが影響を受ける文字列を考慮しなければならない．ある文字に k 回の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 NAG(s,k) で表すと，2 つの文字列間で共有する gram 数の下限値 (=T) は次のように表すことができる．

$$T=\max\{|VG(Q)|-NAG(Q,k),|VG(s)|-NAG(s,k)\} \quad (4)$$

3. Suffix Tree を用いた可変長 N-gram

VGRAM では，gram 辞書を求めるために，gram の最長長 N_{max}，gram の最短長 N_{min}，頻度の閾値 τ の 3 つのパラメータを定める必要があり，そのチューニングコストは大きくなってしまうことが問題である．そのため，筆者らは用いるパラメータを少なくし，かつ検索にかかる時間が VGRAM と同等になることを目的とした．本章ではまず，我々の提案する文字列データから可変長 N-gram を抽出する手順について説明する．その後，可変長 N-gram を用いた類似文字列検索での T-occurrence を解くために必要な $NAG(s,k)$ の求

Vol.2011-DBS-152 No.13
Vol.2011-IFAT-103 No.13
2011/8/3

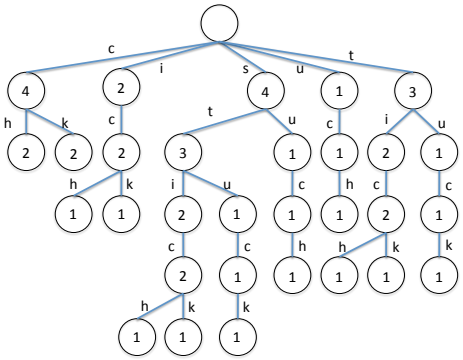

図 1 文字列データの Suffix Tree
Fig. 1 Suffix Tree of strings

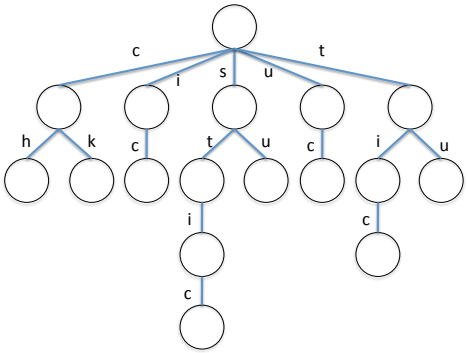

図 2 可変長 N-gram の木構造表現
Fig. 2 Variable length N-gram using Suffix Tree

め方について述べる．

3.1 可変長 N-gram 抽出手法

具体的には suffix tree を用いて可変長の gram の抽出を行った．sugffix tree はテキスト中の部分文字列を探索するのに用いられる手法であるが，その特徴として，テキスト中の全ての長さの gram が表現されるということが挙げられる．筆者らはこの特徴を利用して，可変長の gram の抽出を行うことを考えた．

Suffix Tree を用いた可変長 N-gram は以下の 3 つの手順で構成される．

(1) データ内の各文字列の長さ 2 以上の全ての suffix を同じ 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) 深さが 2 以上の残った枝を連結したものを gram として抽出し，gram 辞書に登録する．

この手法で表 1 の文字列から可変長 N-gram を構築することを考える．このとき Tree の枝刈りに用いる頻度の閾値は 2 であるとする．まずは，各文字列を suffix tree に登録していく．文字列 $id = 0$ の文字列'stick' の長さ 2 以上の suffix は { 'stick', 'tick', 'ick', 'ck'} となる．suffix 'stick' を tree 上に登録し，各ノードの出現頻度を 1 ずつ増やす．このようにすることで，gram 'stic','sti','st' の登録まで行うことができる．同様にして残りの文字列を同じ suffix tree に登録をすると図 1 のようになる．各枝が文字に対応しており，また各ノード内の数字がそのノードの出現頻度，つまりルートからその文字にたどりつく道の枝の各文字を連結させた gram がデータ中に何回出現したかを表している．. 次に枝刈りを行うが，このとき枝刈りの閾値 $\tau = 2$ であるので，tree を深さ優先で探索していき，出現頻度が 2 より少ないノードの枝刈りを行う．枝刈りを行った結果の図を 2 に示す．そして，gram の抽出を行う．gram の抽出は，ルートノードから子ノードをたどっていき，末端のノードまで連結したものを gram とする．また抽出された gram の長さ 2 以上の prefix までを可変長の gram として gram 辞書に登録を行う．本手法では結果として gram 辞書 D={'ch','ck','ic','st','sti','stic','su','tic','tu','uc'} が抽出される．

3.2 NAG

文字列 s 中の文字，$s[i]$ に編集操作を加えたときに影響を受ける gram 数を $B[i]$ の求め方を説明する．提案する手法で抽出された gram の長さが最長であるものの長さを g_{max} とし，gram 辞書 D によって求められた文字列 s の索引語の集合 $VG(s)$ の各要素を (p, g) で表すとする．p は gram の開始位置，g は索引語となる gram である．

簡単のために編集距離が 1 のときについて述べる．このとき $B[i]$ を求めるために以下の 3 つの場合を考える．

Case.1 gram(p, g) に文字 $s[i]$ が含まれる
Case.2 gram(p, g) に文字 $s[i]$ が含まれないが，この編集操作で gram(p, g) が影響を受ける可能性がある
Case.3 gram(p, g) はこの編集操作により全く影響を受けない

Vol.2011-DBS-152 No.13
Vol.2011-IFAT-103 No.13
2011/8/3

[Case.1] gram(p, g) に文字 $s[i]$ が含まれるのは，位置 p が位置 i 以前にあり，gram g の最後の文字位置が位置 i 以降にあるときである．つまり $p \leq i < p + |g| - 1$ を gram(p, g) が満たすときである．このとき，文字 $s[i]$ に編集操作を行うと gram(p, g) は必ず影響を受けるので，このような gram が見つかれば $B[i]$ の値を 1 増やす．

[Case.2] $p < i$ である場合と $p > i$ である場合に分けて考える．$p < i$ であるとき，(p, g) が文字 $s[i]$ への編集操作で影響を受ける可能性があるのは，gram 辞書中の最長の gram 長が g_{max} であることから，p が i から g_{max} 以上離れておらず，[Case.1] に含まれないときである．すなわち，$i - g_{max} + 1 \leq p < p + |g| - 1 \leq i - 1$ を p が満たすときである．このとき，$a = \max(1, i - g_{max} + 1)$ を満たす整数 a に関して，部分文字列 $s[a; i - 1]$ を prefix としてもつ gram が gram 辞書 D 中に存在していなければならない．なぜならば，索引語を生成するときに，ある gram の部分文字列となるような gram では索引付けを行わないからである．a を $a = p$ からはじめ，部分文字列 $s[a; i - 1]$ が gram 辞書 D の中で見つかるか，$a = i - 2$ となるまで a を 1 ずつインクリメントしていく．部分文字列 $s[a; i - 1]$ が gram 辞書 $D(S)$ の中で見つかったときのみ，$B[i]$ の値を 1 増やす．

次に $p > i$ である場合を考える．$p < i$ のときと同様に考えると，(p, g) が文字 $s[i]$ への編集操作で影響を受ける可能性があるのは，$i + 1 \leq p < p + |g| - 1 \leq i + g_{max} - 1$ のときである．このとき $b = \min(|s|, i + g_{max} - 1)$ を満たす整数 b に関して，部分文字列 $s[i + 1; b]$ を suffix としてもつ gram を gram 辞書から探す．このような部分文字列を探すために，gram 辞書中の全ての gram を反転させた逆順の gram 辞書を木構造で保持しておく．そして，b を $b = i + p$ からはじめ，部分文字列 $s[i + 1; b]$ を反転させたものが木構造の中で見つかるか，$b = i + 2$ となるまで b を 1 ずつデクリメントしていく．部分文字列 $s[i + 1; b]$ を反転させたものが木構造の中で見つかったときのみ，$B[i]$ の値を 1 増やす．

[Case.3] gram(p, g) が編集操作により全く影響をうけないのは，gram(p, g) が [Case.1] と [Case.2] のどちらでもない場合である．つまり，p が $p < i - g_{max} + 1$ または $p + |g| - 1 > i + g_{max} - 1$ を満たすとき，gram(p, g) は文字 $s[i]$ への編集操作により影響を受けない．

以上のようにして，$B[i]$ を求めることができる．$B[i]$ が求まれば，その最大値が編集距離 1 のときに影響を受ける gram の最大数となる．よって次式が成り立つ．

$$NAG(s, 1) = \max_i B[i] \tag{5}$$

パラメータ (N_{min}, N_{max}, T)	クエリ検索に要した時間(相対値)
(2, 4, 200)	1.02
(2, 4, 250)	1.00
(2, 4, 300)	1.04
(2, 5, 100)	1.04
(2, 5, 150)	1.01
(2, 5, 200)	1.09

(a) English

パラメータ (N_{min}, N_{max}, T)	クエリ検索に要した時間(相対値)
(2, 4, 200)	1.08
(2, 4, 250)	1.02
(2, 4, 300)	1.03
(2, 5, 100)	1.01
(2, 5, 150)	1.00
(2, 5, 200)	1.02

(b) Spanish

図 3 VGRAM におけるパラメータチューニング
Fig. 3 Tuning parameters in VGRAM

パラメータ T	クエリ検索に要した時間(相対値)
100	1.02
150	1.00
200	1.04
250	1.04
300	1.01

(a) English

パラメータ T	クエリ検索に要した時間(相対値)
100	1.10
150	1.08
200	1.00
250	1.02
300	1.14

(b) Spanish

図 4 提案手法におけるパラメータチューニング
Fig. 4 Tuning parameters in proposed method

4. 評価実験

まずは実験を行うシステムについて説明する．実験を行うシステムの概要は次のとおりである．

(1) 検索の対象となるデータ集合から gram 辞書を作成し，データ内の各文字列の索引付

Vol.2011-DBS-152 No.13
Vol.2011-IFAT-103 No.13
2011/8/3

けを行い，転置索引を構築する

(2) クエリを gram で分割し，各 gram を索引語とする転置索引中のレコードに問い合わせを行い，文字列 id のリストを得る

(3) 得られた文字列のリストをマージし，T-occurrence Problem を解くことで解候補の絞り込みを行う

(4) 解候補中の各文字列とクエリの実際の編集距離を計算し，編集距離の閾値が閾値 (k) 以下のものを検索結果として提示する

実験に用いたデータは SISAP[16] (International Workshop on Similarity Search and Applications) が提供する英語とスペイン語の単語辞書を用いた．各単語辞書にはそれぞれ 69,069 語，86,061 語の単語から構成され，各単語辞書の単語の平均長はそれぞれ 9.4 文字，10.9 文字である．これらの単語辞書を用いて gram 辞書を作成する．

またクエリは，各単語辞書から任意に選んだ 10,000 語の単語を編集距離 1 の文字列に変換したものを用いた．

まずは提案手法，VGRAM のそれぞれで最適なパラメータを求め，最もよい検索性能を示したパラメータを用いて性能を比較した．性能の比較では，T-occurrence を解くのに要した時間，すなわちリストのマージに要した時間，解候補の数，索引付けをし，クエリを問い合わせ結果を提示するのにかかった時間の 3 つで比較を行った．

4.1 パラメータチューニング

VGRAM での最適なパラメータを実験により求めた．各言語ごとで結果は次のようになった．図 3 から分かるように検索に要した時間が最も短かったのは英語のときが $(N_{min}, N_{max}, \tau) = (2, 4, 250)$，スペイン語のときが $(N_{min}, N_{max}, \tau) = (2, 5, 150)$ であった．よって以後の実験ではこの値を用いる．

次に提案手法でもパラメータのチューニングを行った．図 4 から分かるように検索に要した時間が最も短かったのは英語，スペイン語ともに $T = 200$ のときであった．よって以後の性能比較の実験ではこの値を用いる．

4.2 性 能 比 較

性能の比較では，今求めたそれぞれのパラメータを用いて，T-occurrence を解くのに要した時間，解候補の数，索引付けをし，クエリを問い合わせてから結果を提示するのにかかった時間の 3 つで比較を行った．時間の単位にはそれぞれ秒 (s) を用い，フィルタリングされた検索候補の数は 1 クエリあたりの量で示し，単位は「個」である．これは，1 クエリあたり何回の編集距離の計算を行ったかを示している．

比較内容	VGRAM	提案手法
検索候補	69.3	69.6
マージ時間	6.061	5.051
検索時間	21.851	21.185

(a) English

比較内容	VGRAM	提案手法
検索候補	28.8	24.3
マージ時間	7.325	5.902
検索時間	37.952	35.715

(b) Spanish

図 5 性能比較
Fig. 5 Performance comparison

結果を図 5 に示す．実験結果から分かるように最もよいと思われるパラメータのときで提案手法は VGRAM と同等の検索時間で検索ができていることが分かる．スペイン語では検索候補の絞り込みが英語のときと比べてうまく行えているが，これはスペイン語のときの方が gram 辞書に登録された gram の種類が多くなったためであると考えられる．このことから，スペイン語の方が文字間のつながりが小さい，つまりある文字が現れたあとに特定のある文字が出現する確率が少ないということが分かる．データの索引付けに要した時間は VGRAM の方がよい結果となった．これは，VGRAM は gram の最長長をパラメータ (N_{max}) を設定しているため，提案手法に比べて gram の平均長が短くなったからであると考えられる．つまり，提案手法に比べて VGRAM の方が固定長の N-gram の N が小さいときの利点である，索引数が少なくなるという影響が大きくなるためであろう．また，リストのマージに要した時間は提案手法の方がよい結果となった．これは，索引付けに要した時間とは逆で，提案手法は gram の最長長をパラメータとして与えていないために VGRAM より長さの長い gram が抽出できたためである．つまり，固定長の N-gram の N が大きいときの利点である，検索候補が少なくなるという影響が大きいと言える．

以上のことから提案手法は既存の VGRAM と同程度の性能が見込めるということができる．

5. おわりに

N-gram ベースの索引付けを用いた類似文字列検索において，我々は N を可変長にすることで検索をより効率的に行うことのできる suffix tree を用いた新しい可変長 N-gram を提案した．可変長 N-gram の先行研究である VGRAM では事前にパラメータを 3 つ定める必要があり，チューニングのコストが課題であったが，本手法ではパラメータを 1 つしか必

Vol.2011-DBS-152 No.13
Vol.2011-IFAT-103 No.13
2011/8/3

要としない．性能の面でも，評価実験において VGRAM と同等の結果を示すことができ，提案手法の有効性を確認することができた．

今後の展望として，suffix tree の構築にコストがかかるという問題があるため，その構築コストを削減する必要性がある．また，より長い gram の特性を活かした gram の抽出とそれを利用した検索手法を考えていきたい．

参 考 文 献

1) Akiko AIZAWA, Atsuhiro TAKASU, Keizo OYAMA, and Jun ADACHI. Record linkage of multi-source databases: Research trends. *NII journal*, Vol.8, pp. 43–51, 20040227.
2) Alexander Behm, Shengyue Ji, Chen Li, and Jiaheng Lu. Space-constrained gram-based indexing for efficient approximate string search. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, pp. 604–615, Washington, DC, USA, 2009. IEEE Computer Society.
3) Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *In SIGMOD*, pp. 313–324, 2003.
4) Marios Hadjieleftheriou and Chen Li. Efficient approximate search on string collections. *Proc. VLDB Endow.*, Vol.2, No.2, pp. 1660–1661, 2009.
5) Shengyue Ji, Guoliang Li, Chen Li, and Jianhua Feng. Efficient interactive fuzzy keyword search. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pp. 371–380, New York, NY, USA, 2009. ACM.
6) 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 *In VLDB*, pp. 325–336, 2005.
7) Masaru Kitsuregawa and Toyoaki Nishida. Special issue on information explosion. *New Generation Computing*, Vol.28, No.3, pp. 207–215, 2010.
8) VladimirI Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. *Problems of Information Transmission*, Vol.1, No.1, pp. 8–17, 1965.
9) 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.
10) Chen Li, Bin Wang, and Xiaochun Yang. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pp. 303–314. VLDB Endowment, 2007.
11) 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 the 15th conference on Computational linguistics - Volume 1*, COLING '94, pp. 611–615, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.
12) Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, Vol.33, pp. 31–88, March 2001.
13) Gonzalo Navarro and Mathieu Raffinot. Compact dfa representation for fast regular expression search. In *Proceedings of the 5th International Workshop on Algorithm Engineering*, WAE '01, pp. 1–12, London, UK, 2001. Springer-Verlag.
14) Gonzalo Navarro and Mathieu Raffinot. *Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002. ISBN 0-521-81307-7. 280 pages.
15) Sunita Sarawagi and Alok Kirpal. Efficient set joins on similarity predicates. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 743–754, New York, NY, USA, 2004. ACM.
16) http://www.sisap.org.
17) Esko Ukkonen. Approximate string matching with q-grams and maximal matches. Technical report, 1991.
18) Ian H. Witten, Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann, May 1999.
19) 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.
20) Xiaochun Yang, Bin Wang, and Chen Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 353–364, New York, NY, USA, 2008. ACM.
21) 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.
22) 北研二, 津田和彦, 獅子堀正幹. 情報検索アルゴリズム. 共立出版, 2002.