1C-1

高速な類似文字列検索アルゴリズム

岡崎　直観 †　　辻井　潤一 †‡

† 東京大学大学院情報学環　　‡ 英国マンチェスター大学
英国国立テキストマイニングセンター

1　はじめに

類似文字列検索は，文字列集合の中から検索クエリ文字列に似ている文字列を見つけるタスクであり，柔軟な辞書引き，スペル訂正，重複レコード検出など，様々なアプリケーションにおいて必須の技術である．本発表では，「文字列の集合 V の中で，検索クエリ文字列 x と類似度が α 以上の文字列を全て見つけ出す操作」を，**類似文字列検索**と定義する．この操作は，V の部分集合 $\mathcal{Y}_{x,\alpha}$ を求める問題として定式化できる．

$$\mathcal{Y}_{x,\alpha} = \{y \in V \mid \mathrm{sim}(x,y) \geq \alpha\} \tag{1}$$

ここで，$\mathrm{sim}(x,y)$ は文字列 x と y の類似度を与える関数（**類似度関数**）である．この問題は，ある検索クエリ文字列 x と集合に含まれるすべての文字列 $y \in Y$ の類似度を $|V|$ 回計算することで解けるが，文字列集合が膨大（例えば数百万オーダー以上の要素数）になると，実用的ではない．本発表では，式 1 を高速に解くための条件式を導出し，類似文字列検索の新しいアルゴリズムを提案する．さらに，評価実験を行い，既存手法である DivideSkip [1] や Locality Sensitive Hashing [2] と提案手法を比較する．

2　提案手法

本発表では，類似度を計算するための文字列の**特徴**を，集合で表現する．文字列の特徴の取り方は任意であるが，今回は一貫して文字 tri-gram を説明に用いる．例えば，文字列「スパゲッティー」は，{‘＄＄ス’, ‘＄スパ’, ‘スパゲ’, ‘ゲッテ’, ‘ッティ’, ‘ティー’, ‘ィー＄’, ‘ー＄＄’} の 9 要素の文字 tri-gram からなる集合で表現される．ここで，文字列の先頭と末尾に ‘＄’ を挿入し，文字列の開始と終了の特徴を表現した．本稿では，文字列を小文字の変数（x など）で表し，文字列を特徴の集合に変換したものを**特徴集合**と呼び，対応する大文字の変数（X など）で表す．

次に，式 1 の必要十分条件と必要条件を考える．紙面の都合で，本稿では類似度関数にコサイン係数を用いた場合の導出を説明する．文字列 x と y を，それぞれ特徴集合 X と Y で表すと，x と y のコサイン係数は，

$$\mathrm{cosine}(X,Y) = \frac{|X \cap Y|}{\sqrt{|X||Y|}}. \tag{2}$$

この定義式を式 1 に代入すると，必要十分条件が得られる．

$$\left\lceil \alpha \sqrt{|X||Y|} \right\rceil \leq |X \cap Y| \leq \min\{|X|, |Y|\} \tag{3}$$

式 3 は，特徴集合 X と Y のコサイン係数が α 以上になるためには，X と Y が少なくても $\left\lceil \alpha \sqrt{|X||Y|} \right\rceil$ 個の要素を共通に持つ必要があることを示している．以降では，この最小共通要素数を τ（$= \left\lceil \alpha \sqrt{|X||Y|} \right\rceil$）で表す．

ところで，式 3 において $|X \cap Y|$ を無視し，不等式を $|Y|$ について解くと，類似文字列検索の必要条件が得られる．

$$\left\lceil \alpha^2 |X| \right\rceil \leq |Y| \leq \left\lfloor \frac{|X|}{\alpha^2} \right\rfloor \tag{4}$$

この不等式は，類似文字列検索を行うときの Y の探索範囲を表現している．類似文字列検索の必要十分条件と必要条件は，類似度関数にダイス係数，ジャッカード係数，オーバーラップ係数を用いても，同様の手順で導出できる．

Require: x: 検索クエリ文字列
Require: α: 類似度閾値
Require: $D = \{D_l\}$: 転置インデックス配列
1: $X \leftarrow$ `string_to_features`(x)
2: $n \leftarrow$ `min_size`(X, α)
3: $N \leftarrow$ `max_size`(X, α)
4: $S \leftarrow$ []
5: **for** $l = n$ to N **do**
6:　$\tau \leftarrow$ `min_overlap`(X, l, α)
7:　$R \leftarrow$ `overlap_join`(D_l, X, τ)
8:　**for all** $r \in R$ **do**
9:　　r を S に追加
10:　**end for**
11: **end for**
12: **return** S

図 1: 類似文字列検索アルゴリズム

これまでの議論から，類似文字列検索は，次のような一般的な手順で実装することができる．

1. 与えられた検索文字列の特徴集合 X と類似度閾値 α から，式 4 を用いて探索すべき $|Y|$ の範囲 $[n, N]$ を求める
2. その範囲内 $l \in [n, N]$ で，最小共通要素数 τ を求め，式 3 を満たす Y を見つける

例えば，検索クエリ文字列 x =「スパゲティー」で，コサイン類似度の閾値 $\alpha = 0.7$ として類似文字列検索を行うことを考える．文字列の特徴を文字 tri-gram で表現すると，$|X| = 8$ である．式 4 から，Y の要素数に関する探索範囲は $4 \leq |Y| \leq 16$ である．この範囲内，例えば $|Y| = 9$ となる文字列を考慮しているとき，式 3 から，類似文字列の必要十分条件，$6 \leq |X \cap Y|$ が得られる．この必要十分条件は，X の tri-gram のうち，少なくても 6 個は Y にも出現しなければならないことを表す．例えば，y =「スパゲッティー」を考えると，$|X \cap Y| = 6$ である．したがって，y は類似文字列検索の解の一つである．実際，x と y のコサイン類似度は，$6/\sqrt{8 \times 9} = 0.707$ である．

図 1 は，この手順をそのままアルゴリズムとして表現したものである．アルゴリズムの 7 行目の関数 `overlap_join`(D_l, X, τ) は，次の τ **オーバーラップ問題**を解いている．

- 特徴空間上で，検索クエリ文字列 X と τ 個以上の要素を共有する文字列 Y を全て見つける

この部分問題を効率的に解くため，特徴空間から文字列を引くための転置インデックス群 $D = \{D_l\}$ を構成する．まず，すべての文字列をユニークな文字列識別番号（SID）に変換しておく．転置インデックス D_l は，特徴 q が与えられると，その特徴 q を含む文字列（ただし，特徴集合のサイズが l のものに限る）の SID の整列済みリストを返すように設計される．これにより，τ オーバーラップ問題は，転置インデックス D_l に “$X[0]$ or ... or $X[K-1]$”（$K = |X|$）という検索要求を与え，得られた転置リストに含まれる SID を全て走査し，出現頻度が τ 回以上の SID を列挙する処理で解ける．

この処理を実装するのは非常に簡単であるが，検索して得られた転置リスト中の全ての SID を走査しなければならないため，効率が悪い．そこで，本研究では以下に示す 2 つの性質を用いて，文字列の候補の生成と枝刈りを行う．

性質 1　要素数が k の集合 X と，要素数が任意の集合 Y がある．要素数が $(k - \tau + 1)$ となる任意の部分集合 $Z \subseteq X$ を考える．もし，$|X \cap Y| \geq \tau$ ならば，$Z \cap Y \neq \phi$ である *[3].*

Require: d: 転置インデックス
Require: X: 検索クエリの特徴集合
Require: τ: 必要なオーバーラップ数
1: X の要素を，$|\mathtt{get}(d, X[k])|$ の昇順に並び替える
2: $M \leftarrow \{\}$
3: **for** $k = 0$ to $(|X| - \tau)$ **do**
4: **for all** $i \in \mathtt{get}(d, X[k])$ **do**
5: $M[i] \leftarrow M[i] + 1$
6: **end for**
7: **end for**
8: $R \leftarrow []$
9: **for** $k = (|X| - \tau + 1)$ to $(|X| - 1)$ **do**
10: **for all** $i \in M$ **do**
11: **if** $\mathtt{binary\_search}(\mathtt{get}(d, X[k]), i)$ **then**
12: $M[i] \leftarrow M[i] + 1$
13: **end if**
14: **if** $\tau \leq M[i]$ **then**
15: i を R に追加
16: i を M から削除
17: **else if** $M[i] + (|X| - k - 1) < \tau$ **then**
18: i を M から削除
19: **end if**
20: **end for**
21: **end for**
22: **return** R

図 2: τ オーバーラップ問題のアルゴリズム

性質 2 要素数が k の集合 Z と，要素数が任意の集合 Y がある．要素数が h となる任意の上位集合 $X \supseteq Z$ を考える．もし，$|Z \cap Y| < \theta$ ならば，$|X \cap Y| < \theta + h - k$ である．

図 2 に，これらの性質を利用した τ オーバーラップ問題の解法を示した．性質 1 より，$|X \cap Y| \geq \tau$ となるためには，$(|X| - \tau + 1)$ 個の転置リストの中に文字列 y の SID が，少なくても 1 回は含まれていなければならない．そこで，2 から 7 行目で $(|X| - \tau + 1)$ 個の転置リストを走査し，τ オーバーラップ問題の解となり得る文字列の SID を収集している．このとき，転置リストに含まれる SID の数が少ないほど，考慮すべき文字列の候補をコンパクトにできるので，1 行目で転置リストの順番を要素数の少ない順に並び替えている．9 行目から 21 行目では，それぞれの候補 SID（i）が残りの転置リスト中に含まれるかどうか，二分探索で調べ，頻度カウンタ $M[i]$ をインクリメントする．このとき，性質 2 を用いて，候補 i がこれから走査する転置リストの全てに含まれた場合の $|X \cap Y|$ の上限値を $M[i] + (|X| - k - 1)$ として求め，この値が τ よりも小さければ，候補 i を枝刈りする．

3 評価実験

Google Web 1T コーパス（LDC2006T13）に含まれる全ての単語ユニグラム（13,588,391 文字列）をデータセットとし，評価実験を行った．1 つの文字列当たりに含まれる文字 tri-gram の数は約 10.3 で，データセット全体には 301,459 種類の文字 tri-gram が含まれる．提案手法のシステムを C++で実装し，転置インデックスの構築には CDB++[*]を用いた．提案手法を実装したライブラリは，SimString[†]として公開している．実験環境は，インテルの Xeon 5140 CPU（2.33 GHz）と 4GB の主記憶を搭載したサーバーで，類似文字列検索に必要なデータベース（601 MB）を 557.4 秒で構築した．

従来手法として，DivideSkip と Locality Sensitive Hashing (LSH) を実装した．LSH では，データ中の全ての文字列を 64 ビットのハッシュ値に変換した．検索文字列のハッシュ値 $h(x)$ とのハミング距離が δ 以内の文字列を高速に見つけ出すため，データ中の全てハッシュ値のビット列をランダムに入れ替え，入れ替え後のハッシュ値を昇順に整列したリストを

[*] http://www.chokkan.org/software/cdbpp/
[†] http://www.chokkan.org/software/simstring/

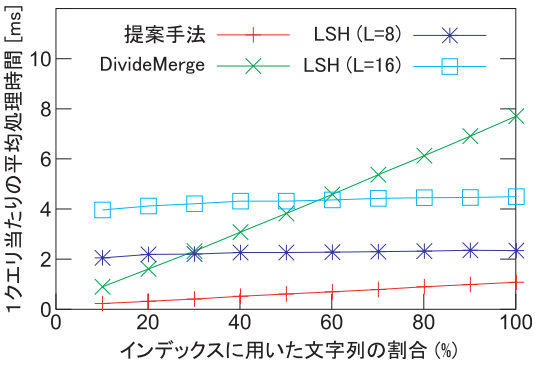

図 3: Google Web 1T コーパスにおけるクエリ処理時間

L 個用意した．このように作られた整列済みハッシュ値リストに二分探索を適用し，探したいハッシュ値（ビット列入れ替え済み）に最も近い文字列，及びその周辺の $B - 1$ 個の文字列を候補とし，ハッシュ値 $h(x)$ とのハミング距離が δ 以内の文字列の中で，類似度が α 以上のものを出力した．ここで，δ，L，B は LSH の類似文字列検索の速度と精度のトレードオフを制御するパラメータであり，$\delta = 16$，$B = 16$ に固定し，$L = 8$ もしくは 16 とした．

図 3 に，データセットの 10%–100%の文字列をインデックスし，同データセットからランダムに選んだ 1,000 件をクエリ文字列として，閾値 $\alpha = 0.8$ で類似文字列検索を行う際の，1 クエリ当たりの平均レスポンス時間を示した．提案手法が最も高速で，100%のデータサイズの時に，1 クエリを 0.99 [ミリ秒/クエリ] で処理していた．図 3 の描画範囲外であるが，クエリ文字列とデータセット中の全ての文字列との類似度を計算するナイーブな実装は，166.1 [秒/クエリ] の処理速度で，提案手法は約 17 万倍高速に類似文字列検索が行える．また，これも図 3 に載せていないが，τ オーバーラップ問題を解く際，性質 1 と性質 2 を利用せず，転置リスト中の SID を全て走査する実装は，403.7 [ミリ秒/クエリ] の処理速度であり，提案手法は 407 倍高速であった．このことから，τ オーバーラップ問題を解く際に，性質 1 や性質 2 で文字列の候補を絞り込んだり，探索途中で枝刈りをすることの効果が伺える．

提案手法は，既存手法の DivideSkip 法よりも約 8 倍高速であった．LSH の処理時間は，約 2.2 [ミリ秒/クエリ]（$L = 8$ のとき），及び約 4.3 [ミリ秒/クエリ]（$L = 16$ のとき）であった．LSH は δ，L，B の値を小さくすることでクエリ処理を高速化できるが，検索の精度（再現率）が低下する．今回の実験条件において，LSH の類似文字列検索の再現率は，36.5%（$L = 8$），40.6%（$L = 16$）であり，常に 100%の再現率が保障される提案手法とは大きな差がある．また，LSH はメモリ使用量が多くなりやすく，今回の実験では，2.6GB（$L = 8$），4.3GB（$L = 16$）の主記憶を消費していた．これに対し，提案手法のメモリ使用量は 601MB であった．以上の実験結果から，提案手法は大規模な類似文字列検索を，実用的な計算量と記憶領域で実現していることが分かった．

参考文献

[1] Chen Li, Jiaheng Lu, and Yiming Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pp. 257–266, 2008.

[2] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, Vol. 51, No. 1, pp. 117–122, 2008.

[3] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. Efficient exact set-similarity joins. In *VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases*, pp. 918–929, 2006.