集合間類似度に対する簡潔かつ高速な類似文字列検索アルゴリズム

岡崎　直観[†]・辻井　潤一[††]

本論文では，コサイン係数，ダイス係数，ジャッカード係数，オーバーラップ係数に対し，簡潔かつ高速な類似文字列検索アルゴリズムを提案する．本論文では，文字列を任意の特徴（tri-gram など）の集合で表現し，類似文字列検索における必要十分条件及び必要条件を導出する．そして，類似文字列検索が転置リストにおける τ オーバーラップ問題として正確に解けることを示す．次に，τ オーバーラップ問題の効率的な解法として，CPMerge アルゴリズムを提案する．CPMerge は，検索クエリ文字列中のシグニチャと呼ばれる特徴と，解候補が枝刈りできる条件に着目し，τ オーバーラップ問題の解候補を絞り込む．さらに，CPMerge アルゴリズムの実装上の工夫について言及する．英語の人名，日本語の単語，生命医学分野の固有表現の３つの大規模文字列データセットを用い，類似文字列検索の性能を評価する．実験では，類似文字列検索の最近の手法である Locality Sensitive Hashing や DivideSkip 等と提案手法を比較し，提案手法が全てのデータセットにおいて，最も高速かつ正確に文字列を検索できることを実証する．また，提案手法による類似文字列検索が高速になる要因について，分析を行う．なお，提案手法をライブラリとして実装したものは，SimString としてオープンソースライセンスで公開している．

キーワード：類似文字列検索，集合間類似度，転置リスト，τ オーバーラップ問題

A Simple and Fast Algorithm for Approximate String Matching with Set Similarity

Naoaki Okazaki[†] and Jun'ichi Tsujii[††]

This paper presents a simple and fast algorithm for approximate string matching in which string similarity is computed by set similarity measures including cosine, Dice, Jaccard, or overlap coefficient. In this study, strings are represented by unordered sets of arbitrary features (e.g., tri-grams). Deriving necessary and sufficient conditions for approximate string, we show that approximate string matching is exactly solvable by τ-overlap join. We propose CPMerge algorithm that solves τ-overlap join efficiently by making use of signatures in query features and a pruning condition. In addition, we describe implementation considerations of the algorithm. We measure the query performance of approximate string matching by using three large-scaled datasets with English person names, Japanese unigrams, and biomedical entity/concept names. The experimental results demonstrate that the proposed method outperforms state-

[†]東北大学大学院情報科学研究科, Graduate School of Information Sciences, Tohoku University
[††]マイクロソフトリサーチアジア, Microsoft Research Asia

of-the-art methods including Locality Sensitive Hashing and DivideSkip on all the datasets. We also analyze the behavior of the proposed method on the datasets. We distribute SimString, a library implementation of the proposed method, in an open-source license.

Key Words: approximate string matching, set similarity, inverted list, t-overlap join

1 はじめに

知的で高度な言語処理を実現するには，辞書，シソーラス，コーパスなどの言語資源の整備・構築が欠かせない．一方，実際のテキストに対して，言語資源を活用するときにボトルネックとなるのが，表層表現が実テキストと言語資源では一致しない問題である．例えば，「スパゲティー」には，「スパゲッティ」「スパゲティ」「スパゲッテー」などの異表記があるが，完全一致の文字列マッチングでは，これらの異表記から言語資源に含まれるエントリ（例えば「スパゲティー」）を引き出すことができない．ウェブなどの大規模かつ統制されていないテキストには，大量の異表記や表記誤りが含まれると考えられ，これらの実テキストに対して言語処理を頑健に適用するには，言語資源とテキストを柔軟にマッチングさせる技術が必要である．

文字列を標準的な表記に変換してからマッチングさせる手法として，ステミング (Porter 1980)，レマタイゼーション (Okazaki, Tsuruoka, Ananiadou, and Tsujii 2008; Jongejan and Dalianis 2009)，スペル訂正 (Brill and Moore 2000; Ahmad and Kondrak 2005; Li, Zhang, Zhu, and Zhou 2006; Chen, Li, and Zhou 2007)，人名表記の照合 (高橋，梅村 1995)，カタカナ異表記の生成及び統一 (獅々堀，津田，青江 1994)，等が代表的である．これらの研究に共通するのは，与えられた文字列から標準形に変換するための文字列書き換え規則を，人手，マイニング，もしくは機械学習で獲得していることである．これらの研究では，語幹やカタカナ異表記など，異表記のタイプに特化した文字列書き換え規則を獲得することに，重点が置かれる．

本論文では，より一般的なタスク設定として，与えられた文字列に似ている文字列を，データベースの中から見つけ出すタスク（**類似文字列検索**）を考える．本論文では，「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記述できる．

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

ただし，$\mathrm{sim}(x, y)$ 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．この問題の単純な解法は，検索クエリ文字列 x が与えられる度に，文字列の類似度を総当たりで $|V|$ 回計算することである．文字列集合の要素数 $|V|$ が小さいときには，総当たりで解を求めることも可能だが，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，実用的な時間で

解けなくなる．本論文では，自然言語処理でよく用いられる類似度関数であるコサイン係数，ジャッカード係数，ダイス係数，オーバーラップ係数に対して，式 1 の簡潔かつ高速なアルゴリズムを提案する．本論文の貢献は，以下の 2 点に集約される．

(1) まず，類似文字列検索における必要十分条件及び必要条件を導出し，式 1 が転置リストにおける τ **オーバーラップ問題** (Sarawagi and Kirpal 2004) として正確に解けることを示す．次に，τ オーバーラップ問題の効率的な解法として，**CPMerge** アルゴリズムを提案する．このアルゴリズムは，τ オーバーラップ問題の解となり得る文字列の数をできるだけコンパクトに保つ特徴がある．提案手法の実装は非常に容易であり，C++で実装したライブラリ[1] を公開している．

(2) 提案手法の優位性を示すため，英語の人名，日本語の単語，生命医学分野の固有表現を文字列データとして，類似文字列検索の性能を評価する．実験では，類似文字列検索の最近の手法である Locality Sensitive Hashing (LSH) (Andoni and Indyk 2008)，SkipMerge, DivideSkip (Li, Lu, and Lu 2008) 等と提案手法を比較する．実験結果では，提案手法が全てのデータセットにおいて，最も高速かつ正確に文字列を検索できることが示される．

本論文の構成は以下の通りである．次節では，類似文字列検索の必要十分条件，必要条件を導出し，式 1 が τ オーバーラップ問題として正確に解けることを示す．第 3 節では，本論文が提案するデータ構造，及び τ オーバーラップ問題の効率的なアルゴリズムを説明する．第 4 節で，評価実験とその結果を報告する．第 5 節では，類似文字列検索の関連研究をまとめる．第 6 節で，本論文の結論を述べる．

2　類似文字列検索の定式化

本研究では，文字列は**特徴**の集合で表現されると仮定する．文字列の特徴の捉え方は，提案手法に依らず任意であるが，本論文では一貫して文字 tri-gram を具体例として用いる．例えば，文字列 $x =$「スパゲッティー」は，9 要素の文字 tri-gram から構成される集合 X で表現される．

$$X = \{\text{‘\$\$ス’}, \text{‘\$スパ’}, \text{‘スパゲ’}, \text{‘ゲッテ’}, \text{‘ッティ’}, \text{‘ティー’}, \text{‘ィー\$’}, \text{‘ー\$\$’}\} \quad (2)$$

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を表現している[2]．一般に，文字数が $|x|$ の文字列 x を文字 n-gram の集合 X で表現したとき，$|X| = |x| + n - 1$ という関係が成り立つ．本論文では，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．$|x|$ を文字列 x の**長さ**，$|X|$ を文字列 x の**サイズ**と呼び，これらを区別する．

[1] SimString: `http://www.chokkan.org/software/simstring/`
[2] この例では，先頭と末尾を表す記号 ‘\$’ を付加したが，これも提案手法に依らず任意である．

なお，特徴に頻度などの重みが付くときは，特徴の識別子を分割することで，重み付きの集合を模擬する．例えば，文字列「トラトラトラ」を文字 tri-gram で表現するとき，‘トラト’と‘ラトラ’が 2 回ずつ出現する．これを集合で表現するには，tri-gram の末尾に出現回数を表す番号を付加すれば良い．これにより「トラトラトラ」は，{‘＄＄ト’#1, ‘＄トラ’#1, ‘トラト’#1, ‘ラトラ’#1, ‘トラト’#2, ‘ラトラ’#2, ‘トラ＄’#1, ‘ラ＄＄’#1 } という集合で表現できる．特徴に出現回数を付与することは実用上重要であるが，説明が冗長になるため，以降では省略する．

本論文では，ダイス係数，ジャッカード係数，コサイン係数，オーバーラップ係数など，集合間のオーバーラップに基づく類似度（集合間類似度）に対して，類似文字列検索アルゴリズムを導出する．文字列の特徴と類似度関数は，類似文字列検索の精度を左右するので，アプリケーションに応じて慎重に選択する必要がある．しかし，どのくらいの精度の類似度関数が必要になるかはアプリケーション依存であるため，文字列の特徴や類似度関数の選び方は本論文の対象外とし，与えられた特徴空間と類似度関数に対して，出来るだけ効率よく $\mathcal{Y}_{x,\alpha}$ を求めるアルゴリズムを提案することに注力する．精細な類似度が必要な場合は，適当な類似度関数に対して緩い閾値 α を用い，提案手法で再現率が高くなるように類似文字列を検索し，関連研究（第 5 節）で紹介する手法などで精査することで，適合率を改善すればよい．

さて，文字列 x と y を，それぞれ特徴集合 X と Y で表すとき，x と y のコサイン係数は，

$$\mathrm{cosine}(X,Y) = \frac{|X \cap Y|}{\sqrt{|X||Y|}}. \tag{3}$$

この定義式を式 1 に代入すると，類似文字列のための必要十分条件が得られる．

$$\left\lceil \alpha\sqrt{|X||Y|} \right\rceil \leq |X \cap Y| \leq \min\{|X|, |Y|\} \tag{4}$$

ここで，$\lceil v \rceil$ は v の整数値への切り上げを表す．また，式 4 には，$|X \cap Y|$ の上限値 $\min\{|X|, |Y|\}$ を不等式として組み込んだ．式 4 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，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．必要十分条件において，$|X \cap Y|$ が取るべき最小の値を，X と Y の**最小オーバーラップ数**と呼び，以降**この数を τ で表す**．τ は，$|X|$，$|Y|$，α に依存して計算される値である．

ところで，式 4 において $|X \cap Y|$ を無視すると，$|X|$ と $|Y|$ に関する不等式を得る．

$$\alpha\sqrt{|X||Y|} \leq \min\{|X|, |Y|\}. \tag{5}$$

この不等式を $|Y|$ について解くと，類似文字列の必要条件が得られる．

$$\left\lceil \alpha^2|X| \right\rceil \leq |Y| \leq \left\lfloor \frac{|X|}{\alpha^2} \right\rfloor \tag{6}$$

ここで，$\lfloor v \rfloor$ は v の整数値への切り捨てを表す．この不等式は，X に対して類似文字列検索を

表 1　集合間類似度を用いた類似文字列検索における $|Y|$ の必要条件，及び $|X \cap Y|$ の必要十分条件

類似度尺度	定義	$\min |Y|$	$\max |Y|$	$\tau = \min |X \cap Y|$
ダイス	$\frac{2|X \cap Y|}{|X|+|Y|}$	$\frac{\alpha}{2-\alpha}|X|$	$\frac{2-\alpha}{\alpha}|X|$	$\frac{1}{2}\alpha(|X|+|Y|)$
ジャッカード	$\frac{|X \cap Y|}{|X \cup Y|}$	$\alpha|X|$	$|X|/\alpha$	$\frac{\alpha(|X|+|Y|)}{1+\alpha}$
コサイン	$\frac{|X \cap Y|}{\sqrt{|X||Y|}}$	$\alpha^2|X|$	$|X|/\alpha^2$	$\alpha\sqrt{|X||Y|}$
オーバーラップ	$\frac{|X \cap Y|}{\min\{|X|,|Y|\}}$	0 (制約無し)	∞ (制約無し)	$\alpha \min\{|X|,|Y|\}$

行う際の，Y に関する探索範囲を表現している．言い換えれば，特徴集合の要素数がこの範囲外の文字列は，無視できる．

なお，同様の導出は，ダイス係数，ジャッカード係数，オーバーラップ係数などの類似度関数に対しても可能である．表 1 に，それぞれの類似度関数の条件式をまとめた．これらの条件式の大元の出典は不明であるが，本論文で導出した条件式は，いくつかの先行研究でも用いられている (Sarawagi and Kirpal 2004; Li et al. 2008; Xiao, Wang, Lin, and Yu 2008b)．

ここで，導出した不等式の利用例を説明する．検索クエリ文字列 $x =$「スパゲティー」とし，コサイン類似度の閾値 $\alpha = 0.7$ で類似文字列検索を行う．また，文字列の特徴を文字 tri-gram で表現することとする（したがって，$|X| = 6+3-1 = 8$ である）．式 6 から，Y の要素数に関する探索範囲は $4 \leq |Y| \leq 16$ である．この範囲内で，例えば $|Y| = 9$ となる文字列を考慮しているとき，式 4 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 1 つである．実際，x と y のコサイン類似度は，$6/\sqrt{8 \times 9} = 0.707$ $(\geq \alpha)$ である．

以上のことをまとめると，種々の類似度関数を用いた類似文字列検索は，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 X と類似度閾値 α から，$|Y|$ の範囲を求める
(2)　その範囲内で，$|X \cap Y|$ の条件を満たす Y を見つける

次節では，これらの手順を効率良く実装するデータ構造とアルゴリズムを議論する．

3　データ構造とアルゴリズム

3.1　データ構造

前節までの議論により，類似文字列検索は次の部分問題を解くことに帰着される．

定義 1 (τ オーバーラップ問題)　検索クエリ文字列の特徴集合 X が与えられたとき，その特徴

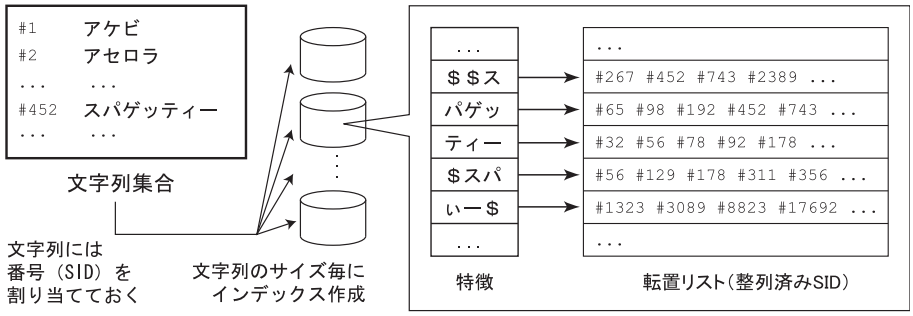


図 1　複数の転置インデックスで構築された類似文字列検索のためのデータ構造

Require: x: 検索クエリ文字列
Require: α: 類似度閾値
Require: $D = \{D_l\}$: 転置インデックスの配列
1: $X \leftarrow \texttt{string\_to\_features}(x)$
2: $n \leftarrow \texttt{min\_size}(X, \alpha)$
3: $N \leftarrow \texttt{max\_size}(X, \alpha)$
4: $R \leftarrow []$
5: **for** $l = n$ to N **do**
6: 　$\tau \leftarrow \texttt{min\_overlap}(X, l, \alpha)$
7: 　$R \leftarrow R \cup \texttt{overlap\_join}(D_l, X, \tau)$
8: **end for**
9: **return** R

図 2　類似文字列検索のアルゴリズム．

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: $R \leftarrow []$
2: $M \leftarrow \{\}$
3: **for all** $q \in X$ **do**
4: 　**for all** $i \in \texttt{get}(d, q)$ **do**
5: 　　$M[i] \leftarrow M[i] + 1$
6: 　　**if** $M[i] = \tau$ **then**
7: 　　　i を R に追加
8: 　　**end if**
9: 　**end for**
10: **end for**
11: **return** R

図 3　AllScan アルゴリズム．

を τ 個以上共有する文字列 Y を全て見つける．

ここで，τ は X と Y の最小オーバーラップ数で，コサイン係数を類似度関数として用いる場合は，$\tau = \left\lceil \alpha\sqrt{|X||Y|} \right\rceil$ である．この部分問題を効率的に解くため，特徴をキーとして，その特徴を含む文字列のリストを値とする連想配列（転置インデックス）を構築する．式 4 から，探索すべき文字列のサイズ $|Y|$ の範囲が絞り込まれること，式 6 から，$|Y|$ に依存して最小オーバーラップ数 τ が決まることを考慮し，文字列のサイズ l 毎に転置インデックス D_l を構築する．また，アルゴリズムを効率よく実行するため，文字列をユニークな文字列識別番号（SID）で表現し，転置リストは特徴を含む文字列の SID を昇順に並べたものを格納することとする．図 1 に，データ構造の実現例を示した．例えば，‘＄＄ス’を特徴に持つ文字列の SID は，#267, #452, #743, #2389, ... であり，「スパゲッティー」の SID は#452 である．図 1 では，文字列のサイズ毎にハッシュ表を構築しているが，SQL などの関係データベースを用いても，同様のデータ構造が実現できる．

図 2 に，類似文字列検索の擬似コードを示す．文字列のサイズ l 毎に構成された転置インデックスの配列 $D = \{D_l\}$ に対して，検索文字列 x，類似度閾値 α が与えられると，この擬似コードは x との類似度が α 以上の文字列の SID のリスト R を返す．１～３行目で，クエリ文字列 x を特徴集合 X に変換し，考慮すべき文字列のサイズの範囲を表 1 から求める．探索範囲内のそれぞれの長さ $l \in [n, N]$ に対し（５行目），最小オーバーラップ数 τ を求め（６行目），`overlap_join` 関数で τ オーバーラップ問題を解き，解集合 R を更新する（７行目）．

3.2　τ オーバーラップ問題のアルゴリズム

3.1 節では，特徴をキーとして，その特徴を含む文字列（SID）のリストを返す転置インデックスを構築した．特徴 q の転置リストに含まれている文字列は，特徴 q を含むことが保証されている．したがって，特徴 $q \in X$ に対応する $|X|$ 個の転置リストの中で，ある文字列 y が c 個の転置リスト 2 回出現するならば，$|X \cap Y| = c$ である．ゆえに，転置リスト上において τ 回以上出現する SID を見つけることで，τ オーバーラップ問題を解くことができる．図 3 に，このアイディアに基づく τ オーバーラップ問題の解法（**AllScan アルゴリズム**）を示した．４行目の関数 `get`(d, q) は，転置インデックス d の中で特徴 q に対応する転置リスト（SID のリスト）を返す関数である．この擬似コードは，転置インデックス d，特徴集合 X，最小オーバーラップ数 τ を受け取り，SID の出現頻度，すなわち $|X \cap Y|$ を連想配列 M に格納し，その値が τ に到達した SID をリスト R に入れて返すものである．表 2 は，検索クエリ文字列 x =「スパゲティー」に対して，Web 日本語 N グラムコーパスのユニグラムの中で，文字数が７（つまり $|Y| = 9$）の文字列を実際に検索するとき，$|X \cap Y|$ の高い文字列 10 件を示したものである（文字列の特徴は tri-gram で表現）．コサイン係数が 0.7 以上の文字列を探すには，$\tau = \left\lceil \alpha\sqrt{|X||Y|} \right\rceil = 6$ であるから，類似文字列検索の解は「スパッゲティー」「スパゲティーニ」「スパゲティー・」「スパゲッティー」の４つである．

AllScan アルゴリズムの実装は簡単であるが，検索に用いる特徴が数多くの文字列に出現するとき，走査する SID の数が非常に大きくなるという欠点がある．例えば，‘ティー’（「ティー」を含む）や ‘ー＄＄’（「ー」で終わる）などの文字 tri-gram は，日本語の多くの語で出現するため，転置リストが大きくなる傾向にある．表 3 に，Web 日本語 N グラムコーパスにおいて，「スパゲティー」の各 tri-gram の転置リストのサイズ（すなわち，各 tri-gram を含む文字列の数）を示した．この表によると，AllScan アルゴリズムは 30,584 種類，35,964 個の SID を走査することになるが，その中でたった 4 個（0.013%）しか解にならない．

走査すべき SID の数を減らすため，τ オーバーラップ問題に関する次の性質に着目する (Arasu, Ganti, and Kaushik 2006; Chaudhuri, Ganti, and Kaushik 2006)．

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

表 2　検索文字列 x=「スパゲティー」に対し，Web 日本語 N グラムコーパス中で $|X \cap Y|$ の大きい文字列（$|Y| = 9$ のとき）

文字列（y）	$\lvert X \cap Y \rvert$
スパッゲティー	6
スパゲティーニ	6
スパゲティー・	6
スパゲッティー	6
チャパゲティー	5
スパゲッチィー	5
スパケッティー	5
セレンゲティー	4
スリムボティー	4
スピンシティー	4

表 3　Web 日本語 N グラムコーパス中で，検索文字列 x=「スパゲティー」の各 tri-gram を含む文字列の数（$|Y| = 9$ のとき）

tri-gram（q）	文字列数（$\lvert$get(d, q)$\rvert$）
＄＄ス	8,293
＄スパ	414
スパゲ	19
パゲテ	5
ゲティ	17
ティー	1,331
ィー＄	874
ー＄＄	24,911
走査する文字列数	35,864
文字列の異なり数	30,584

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．

この性質は, その対偶を考えれば明白である．すなわち, $Z \cap Y = \phi$ ならば, Z の定義から $|X \setminus Z| = k-(k-\tau+1) = \tau-1$ であるので, $|X \cap Y| = |\{(X \setminus Z) \cup Z\} \cap Y| = |(X \setminus Z) \cap Y| + |Z \cap Y| \leq \tau-1$. ゆえに，$|X \cap Y| < \tau$ が示される[3].

性質 1 の利用例を, 先の類似文字列検索を用いて説明する．検索クエリ文字列 x =「スパゲティー」に対し，$|Y| = 9$ かつ，$\tau = 6 \leq |X \cap Y|$ という条件を満たす文字列 y を検索している．検索される文字列 y が $|X \cap Y| \geq 6$ を満たすならば，特徴集合 X 中の任意の $(8-6+1) = 3$ 要素で構成された任意の部分集合 $Z \subset X$ に対し，$Z \cap Y \neq \phi$ である．言い換えれば，特徴集合 X 中の任意の 3 要素を選ぶと，対応する転置リストに，類似文字列検索の解が（あるとすれば）必ず含まれている．この性質を用いると，類似文字列検索の解の候補を絞り込むことができる．解候補を生成するために用いる要素は，シグニチャ (Arasu et al. 2006) と呼ばれる．

では，検索文字列の特徴集合の中で，どの要素をシグニチャとして採用すれば良いのだろうか？　シグニチャの特徴数は性質 1 から決定されるが，その選び方は任意である．したがって，転置リストのサイズが小さい特徴をシグニチャとして採用すれば，解候補生成時に走査する SID の数を減らすことができる．すなわち，文字列データベース中で稀に出現する tri-gram を優先的にシグニチャとして採用し，解の候補を絞り込めばよい．表 3 の例では，「パゲテ」「ゲティ」

[3]二項演算子 $\setminus$ は差集合を表す．つまり $A \setminus B$ は集合 A から集合 B に属する要素を間引いて得られる集合である．

```
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，|get(d, X[k])| の昇順に並び替える
 2: M ← {}
 3: for k = 0 to (|X| − τ) do
 4:     for all i ∈ get(d, X[k]) do
 5:         M[i] ← M[i] + 1
 6:     end for
 7: end for
 8: R ← []
 9: for k = (|X| − τ + 1) to (|X| − 1) do
10:     for all i ∈ M do
11:         if binary_search(get(d, X[k]), i) then
12:             M[i] ← M[i] + 1
13:         end if
14:         if τ ≤ 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) < τ then
18:             i を M から削除
19:         end if
20:     end for
21: end for
22: return R
```

図 4　CPMerge アルゴリズム

```
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，|get(d, X[k])| の昇順に並び替える
 2: M ← []
 3: for k = 0 to (|X| − τ) do
 4:     W ← []
 5:     P ← get(d, X[k])
 6:     m ← 0; p ← 0
 7:     while m < |M| or p < |P| do
 8:         if m = |M| or (p < |P| and M[m].id > P[p]) then
 9:             (w.id, w.freq) ← (P[p], 1)
10:             p ← p + 1
11:         else if p = |P| or (m < |M| and M[m].id < P[p]) then
12:             (w.id, w.freq) ← (M[m].id, M[m].freq)
13:             m ← m + 1
14:         else
15:             (w.id, w.freq) ← (M[m].id, M[m].freq + 1)
16:             m ← m + 1; p ← p + 1
17:         end if
18:         w を W に追加
19:     end while
20:     M ← W
21: end for
22: R ← []
23: for k = (|X| − τ + 1) to (|X| − 1) do
24:     W ← []
25:     for all m ∈ M do
26:         if binary_search(get(d, X[k]), m.id) then
27:             m.freq ← m.freq + 1
28:         end if
29:         if τ ≤ m.freq then
30:             m.id を R に追加
31:         else if m.freq + (|X| − k − 1) ≥ τ then
32:             m を W に追加
33:         end if
34:     end for
35:     M ← W
36: end for
37: return R
```

図 5　CPMerge-opt の擬似コード

「スパゲ」をシグニチャとして選択することになる．

シグニチャによる解候補生成を採用したアルゴリズムを，図 4 に示す．性質 1 より，特徴集合 X を，要素数 $(|X| - \tau + 1)$ のシグニチャ S と，残り L $(= X \setminus S)$ に分解する．このアルゴリズムは，2 から 7 行目で類似文字列検索の解候補をシグニチャ S から獲得し，8 行目から 21 行目で解候補の検証と枝刈りを L で行う．このアルゴリズムは，解候補の生成と枝刈りをしながら転置リストをマージしていくので，**CPMerge アルゴリズム**と命名した．

1 行目で, 検索文字列の特徴集合の要素を, 転置リストのサイズ (要素数) の昇順に並び替える．このとき，$X[k]$ の転置リストの内容をすべてメモリに読み込まなくても，$|\mathtt{get}(d, X[k])|$ の値を取得し，X の要素を並び替えられるようにしておくことは，実用上重要である（この理由は 4.4 節で明らかになる）．特徴集合 X の要素を並び替えたとき，稀な特徴の順番に $X[0]$，..., $X[|X| - 1]$ とアクセスできるものとする．シグニチャ S として採用されるのは，$X[0]$，..., $X[|X| - \tau + 1]$ である．アルゴリズムの 2 から 7 行目では，シグニチャの特徴を持つ文字列をデータベース d

表 4　Web 日本語 N グラムコーパスにおいて，検索文字列 x=「スパゲティー」に対し，$6 \leq |X \cap Y|$ となる文字列 y を見つける際の，候補生成と検証プロセスの実行例（$|Y| = 9$ の場合）

k	0	1	2	3	4	5	6	7	
	候補生成用 tri-gram			検証用 tri-gram					
候補文字列	パゲテ	ゲティ	スパゲ	$スパ	ィー$	ティー	$$ス	−$$	解
アニスパゲスト			○	×.					
イカスパゲティ		○		×.					
エキスパゲージ			○	×.					
クリームスパゲ			○	×.					
ゲティ・ヴィラ		○		×.					
ゲティスパーク		○		×.					
ゲティスパーグ		○		×.					
ゲティズバーグ		○		×.					
ゲティセンター		○		×.					
サーモゲティア		○		×.					
スープ・スパゲ			○	×.					
スパゲッチィー			○	○	○	×.			
スパゲッチパブ			○	○	×.				
スパゲッチュー			○	○	×.				
スパゲッッティ			○	○	×.				
スパゲッテーニ			○	○	×.				
スパゲッティー			○	○	○	○	○	○.	○
スパゲッティイ			○	○	×.				
スパゲッティバ			○	○	×.				
スパゲッテイー			○	○	×.				
スパゲッテリア			○	○	×.				
スパゲットーニ			○	○	×.				
スパゲティー・	○	○	○	○	×	○	○.		○
スパゲティーニ	○	○	○	○	×	○	○.		○
スパッゲティー		○		○	○	○	○	○.	○
セレンゲティ・		○		×.					
セレンゲティー		○		×.					
ターゲティング		○		×.					
チャパゲティー	○	○		×	○	○	×.		
ハムスパゲティ	○	○	○	×	×	×.			
マヨスパゲティ	○	○	○	×	×	×.			
ラスアルゲティ		○		×.					
$P_k = \lvert\texttt{get}(d, X[k])\rvert$	5	17	19	414	874	1,331	8,293	24,911	
$C_k =$（候補文字列数）	5	17	32	17	8	5	2		

から検索し，その転置リストにおける出現回数を連想配列 M に記録する．

先の例と同じ類似文字列検索（$x =$「スパゲティー」，$|Y| = 9$，$\tau = 6 \leq |X \cap Y|$）に対して，CPMerge アルゴリズムの動作例を表 4 に示した．候補生成フェーズでは，「パゲテ」「ゲティ」「スパゲ」の tri-gram を含む文字列を検索し，検索された文字列を解候補とするとともに，該当する箇所に「○」を記している．シグニチャから獲得される解候補の数は 32 で，AllScan アルゴリズムと比べると，解候補数を 0.105%まで絞り込んだことになる．アルゴリズムの 9 行目から 21 行目では，それぞれの候補文字列が，残りの特徴 L を持っているかどうかを調べる．それぞれの解候補 $i \in M$ が（10 行目），特徴 $X[k]$ を持っているかどうかを，転置リスト $\texttt{get}(d, X[k])$ 上における二分探索で調べ（11 行目），転置リストが i を含んでいれば，頻度カウンタをインクリメントする（12 行目）．もし，頻度カウントが τ に到達したら（14 行目），i を結果リスト R に追加し（15 行目），候補 M から削除する（16 行目）．もし，頻度カウントが τ に到達していない場合は，以下の性質を利用して枝刈りの可能性を調べる．

性質 2　要素数が g の集合 X と，要素数が任意の集合 Y がある．要素数が h のある部分集合 $Z \subseteq X$ を考える．もし，$|Z \cap Y| = \theta$ ならば，$|X \cap Y| \leq \theta + g - h$ である．

Z の定義により $|X \setminus Z| = g - h$ であるから，$|(X \setminus Z) \cap Y| \leq g - h$．したがって，この性質は $|X \cap Y|$ の上限値が $(\theta + g - h)$ になることを表現している．図 4 のアルゴリズムでは，$\theta = M[i]$，$g = |X|$，$h = (k+1)$ とおき，$|X \cap Y|$ の上限値を $(M[i] - |X| - k - 1)$ と計算し，この値が τ を下回っているならば（17 行目），候補 i を枝刈りする（18 行目）．

表 4 は，検証フェーズ（$3 \leq k \leq 7$）の動作例も示している．$k = 3$ では，32 個の候補文字列のそれぞれに対して，414 個の SID を含む転置リスト上で二分探索を行い，「＄スパ」という tri-gram を含むかどうか調べている．候補文字列が特徴を含む場合は「○」，含まない場合は「×」が記される．もし，候補文字列が「＄スパ」という tri-gram を含んでおらず，これまでの出現頻度が 1 回だった場合は，今後 $4 \leq k \leq 7$ の全ての転置リストに出現しても，出現頻度の最大値は 5 に留まる．つまり，$|X \cap Y| < 6$ となることが確定しているので，「アニスパゲス」「イカスバゲティ」などの文字列は，$k = 3$ において枝刈りする．表 4 では，枝刈りされる候補に「**×.**」を記している．枝刈りにより，$k = 3$ において 15 個の解候補が枝刈りされ，候補は 17 文字列に減る．$k = 4, 5$ でも同様の処理を行い，解の候補はそれぞれ 8 個，5 個まで絞り込まれる．$k = 6$ では，「スパゲティー・」と「スパゲティーニ」の出現回数が 6 に到達するので，候補集合から解集合に移動させる（**○.** で表示）．$k = 7$ では，「スパゲッティー」と「スパッゲティー」の出現回数が 6 に到達し，全ての候補の検証が終了したことになる．

CPMerge アルゴリズムにおいて，$X[k]$ の転置リストを処理した後に残る解候補の数を C_k，$X[k]$ の転置リストの要素数を $P_k = |\mathtt{get}(d, X[k])|$ とする．CPMerge の検証フェーズでは，それぞれの候補に対して二分探索を行うため，9 行目の各 k に対して，10〜20 行目の計算量は $O(C_{k-1} \log P_k)$ である．$X[k]$ の並び順の定義から，k が大きくなると P_k も増加するが，枝刈りが有効に働けば，C_k が小さくなる．表 4 の例では，各 k に対して $C_{k-1} \log P_k$ の値は，193 $(k = 3)$，115 $(k = 4)$，57.5 $(k = 5)$，45.1 $(k = 6)$，20.2 $(k = 7)$ であり，9〜21 行目のループが進むにつれて，計算量の見積りが減少する．検索クエリ文字列やデータベースの文字列集合の tri-gram の分布により，C_k や P_k の傾向が異なるので，計算量の見積を一般的に行うことは難しい．そこで，第 4 節では，CPMerge アルゴリズムが実際のデータセットに対して動作する際の，解の候補数，転置リストに含まれる SID の数などの統計情報を報告する．

3.3　実装上の工夫

図 4 のアルゴリズムでは，SID をキーとして頻度を格納する連想配列 M を用いていた．実は，転置リストが整列済みの SID で構成されるという性質を利用すれば，情報検索における転置リストのマージ (Manning, Raghavan, and Schütze 2008) と同様に，連想配列をリスト構造（可変

長配列）で代用できる．主要なプログラミング言語では連想配列を容易に扱えるが，アクセスのコストがリスト構造よりも大きいので，連想配列をリスト構造で置き換えることで，検索処理の高速化が期待できる．

図 5 は，図 4 から連想配列を排除し，リスト構造のみで CPMerge アルゴリズムを実装するもの（CPMerge-opt）である．図 5 の 2〜21 行目は，図 4 の 2〜7 行目に対応し，解の候補生成を行う．2 行目では，解候補の頻度を計測する変数 M を初期化しているが，その型は連想配列（{}）から，可変長配列（[]）に変更されている．CPMerge-opt では，M の要素は (SID, 頻度) のタプルであり，要素は SID の昇順に並べる．3〜21 行目の基本的な流れは，$(k-1)$ における解候補リスト M と，$P = \texttt{get}(d, X[k])$ の転置リストを，先頭から順に比較していき，一時変数 W に k における解候補リストを作成する．最後に，M を W で上書きし（20 行目），$k+1$ のステップへと進む．各 k において，W を空のリストで初期化し（4 行目），M と P でこれから処理する要素の位置（インデックス）を管理する変数 m と p を，それぞれ 0 で初期化する（6 行目）．7 行目から 19 行目までは，M と P の全ての要素を処理し終わるまで，以下の処理を繰り返す．

(1) もし転置リスト P の SID（$P[p]$）が，$(k-1)$ における解候補リスト M に含まれていない場合（8 行目），$P[p]$ を新しい候補として W に登録し（9 行目），p をインクリメントする（10 行目）．

(2) もし，$(k-1)$ における解候補リスト M 中の SID（$M[m]$.id）が，転置リスト P に含まれていない場合（11 行目），$M[m]$ を W にそのまま追加し（12 行目），m をインクリメントする（13 行目）．

(3) それ以外の場合，すなわち転置リスト P の要素 $P[p]$ と解候補リスト M 中の $M[m]$.id が等しい場合（14 行目），$M[m]$ の頻度をインクリメントしたものを W に追加し（15 行目），p と m の両方をインクリメントする（16 行目）．

図 5 の 22〜36 行目は，図 4 の 8〜21 行目に対応し，解の候補の検証と枝刈りを行っている．CPMerge-opt では，$(k-1)$ における解候補リスト M に対して，転置リスト $\texttt{get}(d, X[k])$ で検証を行い，枝刈りされなかった候補を一時変数 W に待避し，k における処理が終わったら M を W で上書きしている．図 4 と図 5 のその他の箇所は，ほとんど同じである．

4 実験

4.1 比較したシステム

本節では，大規模な文字列データセットに対して，種々の類似文字列検索アルゴリズムの性能を比較し，提案手法の有用性を示す．実験に用いたシステムは，以下の通りである．なお，先行研究の詳細については，5 節を参照されたい．

- **提案手法**: τ オーバーラップ問題を CPMerge（図 4）で解くもの

- **提案手法-opt**: CPMerge を図 5 の擬似コードで高速化したもの
- **総当たり法**: 検索クエリが与えられる毎に，データベース内の全ての文字列と類似度を計算し，閾値以上の文字列を見つけ出す方法
- **AllScan**: τ オーバーラップ問題を AllScan アルゴリズム（図 3）で解くもの
- **Signature**: τ オーバーラップ問題を CPMerge（図 4）で解くが，解候補の枝刈りを行わないもの（図 4 の 17〜18 行目を削除）
- **SkipMerge**: τ オーバーラップ問題を SkipMerge アルゴリズム (Li et al. 2008) で解く
- **DivideSkip**: τ オーバーラップ問題を DivideSkip アルゴリズム (Li et al. 2008) で解く[4]
- **MergeOpt**: τ オーバーラップ問題を MergeOpt アルゴリズム (Sarawagi and Kirpal 2004) で解く．ただし，MergeOpt は重み付きの類似度を用いた類似文字列検索アルゴリズムであり，本論文と実験設定を揃えるため，次のような修正を行った．　文字列の特徴の重みはすべて等しいこととする．　提案手法と同様に，転置リストはサイズの昇順でソートする．　転置リストを候補生成用 S と検証用 L に分割するときは，提案手法と同様に S をシグニチャの転置リストとする．
- **Locality Sensitive Hashing (LSH)** (Andoni and Indyk 2008; Ravichandran, Pantel, and Hovy 2005): 文字列の tri-gram を特徴とし，64 ビットの局所鋭敏ハッシュ（LSH）値を計算する関数を $h(x)$ とする．２つのハッシュ値 v_1 と v_2 の，ビット単位でのハミング距離（ビットの異なり数）を，$\mathrm{hdist}(v_1, v_2)$ と書く．このシステムは，クエリ文字列 x が与えられると，そのハッシュ値 $h(x)$ とのハミング距離が δ 以内の文字列を V から探し，解候補となる文字列集合 C（$|C| \ll |V|$）を求める．

$$C = \{y \in V \mid \mathrm{hdist}\,(h(x), h(y)) \leq \delta\} \tag{7}$$

解候補のそれぞれの文字列 $y \in C$ に対して，実際に x との類似度を計算し，閾値以上の文字列を解とする．

式 7 の正確な解を求めることは難しいため，Ravichandran ら (Ravichandran et al. 2005) の手順を参考に，近似解を求める．基本的なアイディアは，データベース中の文字列のハッシュ値のビット列を並び替え，検索クエリの（ビット列を並び替えられた）ハッシュ値の近傍を探すという試行を繰り返せば，式 7 の近似解が求まるというものである．ある文字列のハッシュ値 $h(y)$ のビット列を，置換 σ_p で並び替えたものを $\sigma_p(h(y))$ で表し，そのような置換をランダムに P 種類用意し，$\Sigma = \{\sigma_p\}$ とする．置換 σ_p を用いて，データベースに含まれている全ての文字列 $y \in V$ のハッシュ値のビット列を並び替え，辞書順にソートしたハッシュ値リストを a_p とする．置換を P 種類別々に適用すると，ハッ

[4] 今回の実験では，パラメータ $\mu \in \{0.01, 0.02, 0.04, 0.1, 0.2, 0.4, 1, 2, 4, 10, 20, 40, 100\}$ を試し，最も検索レスポンスが速かった $\mu = 40$ を採用した．

シュ値のリストも P 種類作られる．

すると，a_p の中でクエリの（置換が適用された）ハッシュ値 $\sigma_p(h(x))$ に近い要素を二分探索で求め，その近傍の W 個の文字列の中でハミング距離が δ 以内のものを見つけ出すことで，式 7 を近似的に求めることができる．この処理を，準備しておいた P 個の置換と，対応する P 個のハッシュ値リストに対して行い，式 7 の近似解の精度を向上させる．

LSH は，類似文字列検索の解の近似解を高速に求める手法であるため，検索漏れ（類似文字列が検索されない状況）が生じることがある．LSH では，ハミング距離の閾値（δ），並び替えハッシュ値リストの数（P），近傍探索の幅（W）の 3 つのパラメータで検索速度と再現率のトレードオフを調整する．今回の実験では，実験的に $\delta = 24$，$P = 24$ と決定し[5]，W を $W \in \{16, 32, 64\}$ と変えながら性能を測定した．

総当たり法と LSH 以外のすべてのシステムは，図 2 の実装を共有しており，τ オーバーラップ問題の解法が性能差となって現れる．ハッシュ・データベースとしては，CDB++[6] を用い，提案手法を C++で実装したライブラリとして，SimString[7] を公開している．すべての実験は，Intel Xeon 5140 CPU（2.33 GHz）と 8GB の主記憶を搭載した Debian GNU/Linux 4.0 のアプリケーション・サーバー上で行った．転置インデックスはファイル上に構築し，実験時には必要に応じて主記憶に読み込んでいる．

4.2 実験に用いたデータセット

実験に用いたデータセットは，以下の 3 つである．

- **IMDB**: IMDB データベース[8]のファイル `actors.list.gz` から抜き出したすべての俳優名（1,098,022 文字列，18MB）．1 つの文字列当たりの文字 tri-gram の特徴数は 17.2，データセット全体における文字 tri-gram の種類数は 42,180 である．SimString は，このデータセットから 83MB のインデックスファイル群を，56.6 秒で構築した．
- **日本語ユニグラム**: Web 日本語 N グラム第 1 版[9]に収録されている単語ユニグラム（2,565,424 文字列，49MB）．1 つの文字列当たりの文字 tri-gram の特徴数は 20.8，データセット全体における文字 tri-gram の種類数は 137,675 である[10]．SimString は，このデータセットから 220M のインデックスファイル群を，134.0 秒で構築した．
- **UMLS**: Unified Medical Language System (UMLS)[11] に収録されている生命医学分野の英語の概念や記述（5,216,323 文字列，212 MB）．評価に用いる文字列は，UMLS Release

[5] パラメータ P は，並び替えたハッシュ値リストが全てメモリ上に載るようにするため，24 と決定した．パラメータ δ として，$\delta \in \{8, 16, 24\}$ を試し，検索速度と再現率のバランスが良かった 24 を採用した．

[6] `http://www.chokkan.org/software/cdbpp/`

[7] `http://www.chokkan.org/software/simstring/`

[8] `ftp://ftp.fu-berlin.de/misc/movies/database/`

[9] `http://www.gsk.or.jp/catalog/GSK2007-C/catalog.html`

[10] 実験では日本語の文字列を UTF-8 で表現し，UTF-8 の 1 バイトを 1 文字とみなして tri-gram を作っている．

[11] `http://www.nlm.nih.gov/research/umls/`

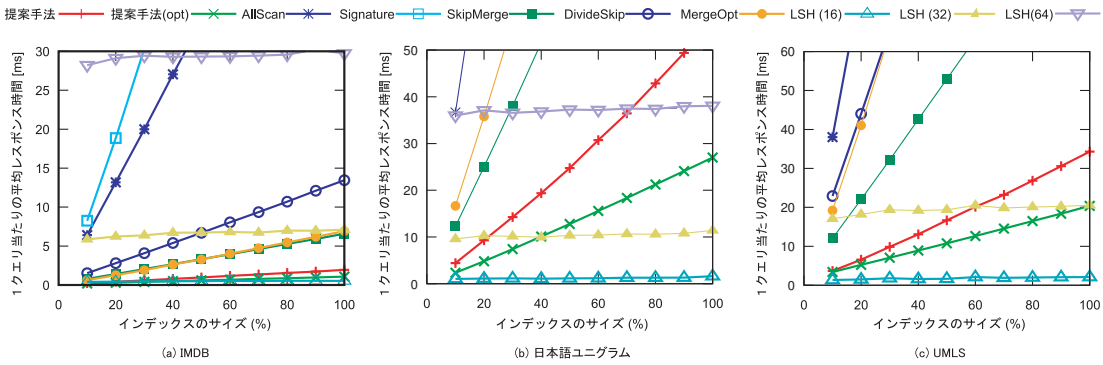


図 6　1 クエリ当たりの平均レスポンス時間（横軸はデータセットのサイズ）

2009AA (April 6, 2009) の `MRCONSO.RRF.aa.gz` 及び `MRCONSO.RRF.ab.gz` というファイルに含まれる全ての英語の概念名である．一つの文字列当たりの文字 tri-gram の特徴数は 43.6，データセット全体における文字 tri-gram の種類数は 171,596 である．SimString は，このデータセットから 1.1GB のインデックスファイル群を，1216.8 秒で構築した．

それぞれのデータセットにおいて，1,000 個の文字列をランダムに選び，テスト用のクエリ文字列とした．完全には一致しない文字列で検索する状況をシミュレートするため，1,000 個の文字列のうち，1/3 の文字列はそのまま，1/3 の文字列には 1 文字をランダムな文字に置換，残りの 1/3 の文字列には 2 文字をランダムな文字に置換している．

4.3　1 クエリあたりの平均レスポンス時間

図 6 に，各データセットでコサイン係数が 0.7 以上の類似文字列を検索するときの，1 クエリあたりの平均レスポンス時間を示した．グラフの横軸は，データベースに登録する文字列の数（$|V|$）で，データセット全体の 10%から 100%まで変化させた．また，表 5 に，各データセットをすべて（100%）利用したときのシステム性能として，検索の再現率（Recall），1 クエリあたりの平均レスポンス時間（Mean），1 クエリに対する最遅レスポンス時間（Max）をまとめた．実験したシステムの中では LSH ($W = 16$) が最も高速に文字列を検索でき，データサイズが 100%の時の平均レスポンス時間は，0.53 ms（IMDB），1.61 ms（日本語ユニグラム），2.11 ms（UMLS）であった．また，平均レスポンス時間に対して最遅レスポンス時間がどのくらい遅くなるのかに着目すると，LSH は入力クエリの影響を受けにくいことが分かる[12]．これは，LSH では探索範囲が δ，P，W などのパラメータで一定に保たれるからである．一方，LSH 以外の手法（総当たり法を除く）は，クエリ文字列に応じて $|Y|$ の探索範囲が変化し，さらに，クエ

[12]LSH (W=16) に関しては，最遅レスポンス時間が平均レスポンス時間と比べてかなり遅くなっているが，これは 1 クエリ当たりの処理時間が非常に短いため，実行時間の測定値の誤差が出やすくなるためと考えられる．

表 5　各データセットを 100%利用したときのシステムの性能（Recall: 再現率，Mean: 平均レスポンス時間 [ms/query]，Max: 最遅レスポンス時間 [ms/query]）．総当たり法に対しては，平均レスポンス時間のみを掲載した．

システム	IMDB			日本語ユニグラム			UMLS		
	Recall	Mean	Max	Recall	Mean	Max	Recall	Mean	Max
提案手法	100.0	1.94	24.1	100.0	56.03	636.8	100.0	34.30	509.4
提案手法-opt	100.0	1.07	11.8	100.0	26.99	275.1	100.0	20.37	334.2
総当たり法	100.0	(32.8×10^3)		100.0	(92.7×10^3)		100.0	(416.3×10^3)	
AllScan	100.0	69.82	159.0	100.0	669.46	1530.0	100.0	355.56	4390.0
Signature	100.0	124.01	2229.5	100.0	6395.76	62480.0	100.0	39139.20	583600.0
SkipMerge	100.0	6.58	55.9	100.0	138.74	949.6	100.0	105.68	1553.1
DivideSkip	100.0	13.46	192.5	100.0	649.51	2460.0	100.0	225.96	4722.2
MergeOpt	100.0	6.92	92.5	100.0	209.61	1880.0	100.0	197.28	2910.0
LSH (W=16)	15.4	0.53	3.0	7.5	1.61	13.5	4.0	2.11	44.5
LSH (W=32)	20.6	7.09	12.3	11.4	11.40	34.1	7.1	20.57	90.6
LSH (W=64)	25.8	29.72	36.7	15.4	38.07	82.8	11.1	79.73	224.3

リ文字列に応じて転置リストのサイズが異なる（つまり，処理すべき SID の数が変化する）ため，レスポンス時間のばらつきが大きくなる．

しかし，LSH ($W = 16$) は検索漏れが非常に多い．総当たり法で検索される類似文字列を正解とみなし，そのどのくらいをカバーできているか（再現率）を測定すると，LSH ($W = 16$) の再現率は，15.4%（IMDB），7.5%（日本語ユニグラム），4.0%（UMLS）であった．これは，式 7 の近似解の精度が悪いためである．LSH の再現率を改善するには，周辺探索の幅（W）を大きくすればよいが，レスポンス時間を犠牲にしなければならない．例えば，LSH（$W = 64$）では，再現率は 25.8%（IMDB），15.4%（日本語ユニグラム），11.1%（UMLS）に改善されるが，レスポンス時間は 29.72 ms（IMDB），38.07 ms（日本語ユニグラム），79.73 ms（UMLS）まで遅くなる．これに対し，提案手法では調整するパラメータもなく，正確な解（100%の再現率）が保証されている．したがって，類似文字列検索の再現率を重視する場合は，提案手法の方が優れている．

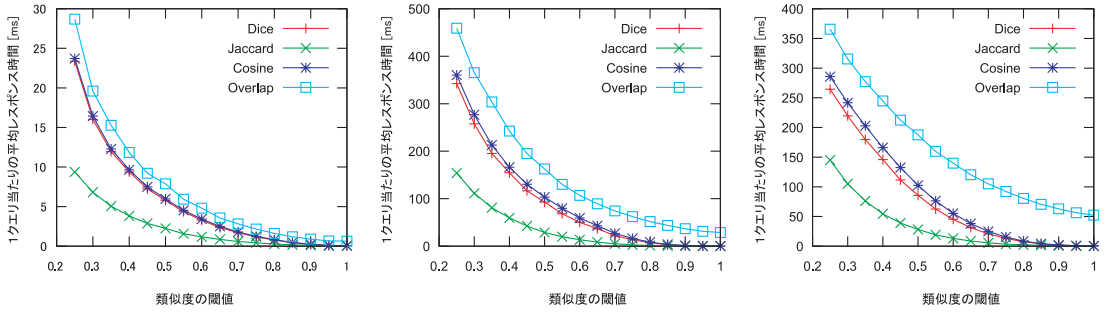
提案手法-opt は，正確な解が得られるシステムの中で最もレスポンスが速く，データサイズが 100%の時の平均レスポンス時間は，1.07 ms (IMDB)，26.99 ms (日本語ユニグラム)，20.37 ms (UMLS) であった．提案手法と提案手法-opt を比較すると，図 5 の実装上の工夫を利用することで，レスポンス時間が 1.7〜2.1 倍高速になった．そこで，以降の説明では単に「提案手法」

というと，図 5 の工夫を適用した「提案手法-opt」を指すこととする．

総当たり法のレスポンス時間は図 6 にプロットできないくらい遅く，32.8 s (IMDB)，92.7 s (日本語ユニグラム)，416.3 s (UMLS) であった．提案手法は，総当たり法よりも 3,400〜20,000 倍高速に動作し，類似文字列検索を実用的な速度で実現している．提案手法は，AllScan アルゴリズムよりもかなり高速に動作し，検索速度は 65.3 倍（IMDB），24.8 倍（日本語ユニグラム），19.2 倍（UMLS）高速であった．提案手法と Signature システムを比較すると，提案手法の方が Signature よりも 115.9 倍（IMDB），237.0 倍（日本語ユニグラム），2323 倍（UMLS）高速であった．Signature システムと提案手法の差は，解候補の枝刈り（図 4 の 17〜18 行目）のみであるが，この処理を省くと大幅にレスポンスが低下し，AllScan アルゴリズムよりも遅くなる．これらのシステムの比較から，τ オーバーラップ問題を解く際に解候補を絞り込んでおくこと，二分探索の回数を減らすために解候補の枝刈りをすることが，非常に重要であることが伺える．先行研究である MergeOpt，SkipMerge，DivideSkip は，各転置リストの先頭（SID の小さい方）から SID を優先順位付きキューに挿入するアルゴリズムを採用しており，τ オーバーラップ問題の解き方が提案手法と全く異なる．このため，レスポンス時間の差の要因を分析することは難しいが，提案手法は MergeOpt よりも 6.47〜9.68 倍，SkipMerge よりも 5.14〜6.15 倍，DivideSkip よりも 11.1〜24.1 倍高速であった．

IMDB データセットにおいて，提案手法が検索に最も時間を要したクエリ文字列は，“Morales, Michael (VIII)” で，11.8 ms を要した．以下，“Reiner, Robert (I)” (9.2 ms)，“Dore, Michael (I)” (9.2 ms)，“Dow, Christopher (III)” (8.6 ms)，“Cain, William (II)” (8.0 ms) と続く．これらのクエリが遅いのは，データセット中に似ている文字列（例えば “Morales, Michael (III)” や “Morales, Rafael (VIII)” など）が多く，τ-オーバーラップ問題を解くときに多くの解候補を抱えるためである．例えば，“Morales, Michael (VIII)” というクエリに対し，データセット中の 72,375 個の文字列が解候補となり，最終的に解になったのは 42 文字列であった．一方，提案手法と SkipMerge アルゴリズムのレスポンス時間の差を計算したとき，提案手法の改善が最も顕著に表れたクエリの上位３件は，“Morales, Michael (VIII)” (-44.4 ms)，“Pittman, Robert (III)” (-39.0 ms)，“Richards, Jon (VII)” (-36.6 ms) であった．ここでも，“Morales, Michael (VIII)” というクエリが登場し，その他の２つのクエリも解候補が非常に多い（40,000 個以上）ことから，データセット中にクエリ文字列と似ている文字列が多く存在するとき，提案手法の優位性が際立つと考えられる．

逆に，提案手法が SkipMerge アルゴリズムよりも遅くなったクエリは無かったものの，改善が全く見られなかったクエリとして，“Zhao,lSh@nqiu” (±0 ms)，“Peral9a,dStacy” (±0 ms)，“Sen]g[renqing” (±0 ms) などが見つかった．これらのクエリは，元々 “Zhao, Shenqiu”, “Peralta, Stacy”, “Senggerenqing” の文字列にノイズが加わったものと考えられるが，転置リストに含まれる文字列の種類数が非常に少なく，それぞれ 3 個，108 個，18 個であった．したがって，転



(a) IMDB　(b) 日本語ユニグラム　(c) UMLS

図 7　1 クエリ当たりの平均レスポンス時間（横軸は類似度閾値）

置リストにおいて処理すべき文字列の数が圧倒的に少ないため，アルゴリズム間の差が出にくくなったと考えられる．このような場合でも，提案手法は SkipMerge アルゴリズムよりも遅くならず，同程度のレスポンス時間を出していた．

図 7 は，異なる類似度関数と閾値を用いたときの，提案手法のレスポンス時間を示している．類似度の閾値を低く設定すると，類似文字列検索の解となる文字列の数 $|\mathcal{Y}|$ が大きくなるので，提案手法のレスポンス時間が遅くなる．類似度関数の良さはタスク依存で決めることであるが，同じ閾値を用いた場合はジャッカード係数が最も速く，ダイス係数とコサイン係数が同程度，オーバーラップ係数が最も遅いという傾向が見られる．この傾向は，類似度関数の性質（どの程度文字列を類似していると見なすか）によって，類似文字列検索の解の数が異なることから説明できる．例えば，日本語ユニグラムコーパスにおいて閾値 0.7 で類似文字列検索を行った場合，ジャッカード係数，ダイス係数，コサイン係数，オーバーラップ係数が返す解文字列数の平均は，それぞれ，1.2 個，14.8 個，16.2 個，1036.4 個であった．したがって，類似文字列検索では，最も多い解を返すオーバーラップ係数が遅く，最も少ない解を返すジャッカード係数が速くなる．また，表 1 の必要条件から求まる $|Y|$ の探索範囲が，ジャッカード係数では最も狭く[13]，オーバーラップ係数では $|Y|$ の探索範囲に制約が付かないことから，類似度関数による検索速度の違いを推察できる．

4.4　提案手法の動作統計

表 6 は，提案手法が各データセットにおいて類似文字列検索を行うときの，様々な統計情報をまとめたものである（類似度にコサイン係数を用い，閾値は 0.7 とした）．この表の見方を IMDB データセットで説明すると，提案手法はサイズが 8.74 から 34.06 までの文字列を検索対

[13] $0 \leq \alpha \leq 1$ であるから，$|Y|$ の探索範囲の下限に関して，$\alpha^2|X| \leq \alpha|X|$，$\frac{\alpha}{2-\alpha}|X| \leq \alpha|X|$ が成立する．$|Y|$ の探索範囲の上限に関しても，$|X|/\alpha \leq |X|/\alpha^2$，$|X|/\alpha \leq \frac{2-\alpha}{\alpha}|X|$ であり，ダイス係数やコサイン係数よりもジャッカード係数の方が，同じ閾値 α を用いたとき，$|Y|$ の探索範囲が狭くなる．

表 6　提案手法が各データセットで類似文字列検索を行うときの動作統計

項目	IMDB	日本語	UMLS	説明
$\min\|y\|$	8.74	10.73	21.87	検索対象となった文字列の tri-gram の最大サイズ
$\max\|y\|$	34.06	41.01	88.48	検索対象となった文字列の tri-gram の最小サイズ
τ	14.13	17.79	47.77	最小オーバーラップ数
$\|\mathcal{Y}\|$	4.63	16.17	111.79	１つのクエリ文字列あたり検索された文字列の数
全体				
$\|X\|$	17.7	23.7	61.1	１つのクエリ文字列に関係する転置リストの数
# SIDs	16,155.1	146,408.0	49,561.4	転置リスト中に存在する SID の数
# SID 種類数	9,788.7	39,884.2	17,457.5	転置リスト中に存在する SID の種類数
解候補の生成				
# 転置リスト	4.6	7.0	14.3	解の候補生成時に使われた転置リストの数
# SIDs	279.7	6,886.0	1,756.3	解の候補生成時に走査した SID の数
# 候補	232.5	3,897.0	1,149.7	一つのクエリに対する解の候補数
解候補の検証				
# 転置リスト	4.3	11.8	17.4	二分探索の対象となった転置リストの数
# SIDs	7,561.8	247,665.0	20,443.7	二分探索の対象となった SID の数

象とし，1クエリあたり4.63文字列を解として返した．提案手法の候補生成フェーズでは，平均4.6個の転置リストに含まれる279.7個のSIDを走査し，232.5個の解候補を得た．提案手法の候補検証フェーズでは，平均4.3個の転置リストに対して二分探索を行い，7,561.8個のSIDが二分探索の対象となった．これに対し，AllScanアルゴリズムは，17.7個の転置リストに含まれる16,155.1個のSIDを走査しなければならず，平均4.63個の解を求めるのに，9,788.7個の文字列を候補として考慮する必要があった．

この表は，提案手法の3つの特筆すべき特徴を表している．

- 提案手法はAllScanアルゴリズムと比較すると，走査するSIDの数を格段に減らしている．例えば，IMDBデータセットにおいて解候補を得るために，AllScanアルゴリズムは16,155.1個のSIDを走査する必要があったが，提案手法は279.7個のSIDを走査するだけで済んだ．別の言い方をすれば，提案手法はAllScanアルゴリズムと比較すると，1.1%〜3.5%の文字列を走査すれば，解候補が得られることを示している．
- 提案手法はAllScanアルゴリズムと比較すると，解候補の数を9,788.7から232.5に減らしている．すなわち，解候補の数は提案手法により1.2%〜6.6%まで削減された．
- 提案手法はAllScanアルゴリズムと比較すると，主記憶上に展開すべき転置リストの数を減らすことができる．提案手法は，τオーバーラップ問題を解くために，8.9 (IMDB)，18.8 (日本語ユニグラム)，31.7 (UMLS) 個の転置リストを使っている[14]．AllScanアルゴリズムが用いる転置リストの数と比べると，提案手法は50.3% (IMDB)，53.6% (日本語ユニグラム)，51.9%（UMLS）の転置リストしかアクセスしないことを意味する．これは，図4のアルゴリズムで，$k \approx 0.5|X|$付近で解候補の検証・枝刈りが完了し，解の候補が0になっているからである．提案手法では，kが大きくなるにつれ，転置リストのサイズが大きくなるが，サイズの大きい転置リストをメモリ上に展開することなく，τオーバーラップ問題を解けるのは，提案手法の大きなアドバンテージである．

5 関連研究

類似文字列検索は，データベースやデータマイニングの分野で，盛んに研究が行われている．その中で最も多い研究は，文字列の編集距離を距離尺度として用いるものである．Gravanoら (Gravano, Ipeirotis, Jagadish, Koudas, Muthukrishnan, and Srivastava 2001) は，n-gram[15]で文字列のインデックスを作り，オーバーラップの個数，位置，文字列のサイズなどで編集距離の制約を満たす解を絞り込む方法を提案した．Kimら (Kim, Whang, Lee, and Lee 2005) は，n-gramが出現した場所をインデックスに効率よく格納するため，2階層のn-gramインデック

[14] これらの値は，4.6 + 4.3，7.0 + 11.8，14.3 + 17.4 として計算される．
[15] データベースの分野ではq-gramと呼ばれることが多い

スを提案した．Li ら (Li, Wang, and Yang 2007) は，クエリの処理速度を向上させるため，可変長の n を用いた n-gram インデックスを用いた．Lee ら (Lee, Ng, and Shim 2007) は，ワイルドカードを含む n-gram でインデックスを作り，編集距離制約の類似文字列を効率よく検索する手法を考案した．Xiao ら (Xiao, Wang, and Lin 2008a) は，検索クエリとマッチングできなかった n-gram を活用する，Ed-Join アルゴリズムを提案した．

文字列を n-gram などで表現することなく，編集距離に基づく類似文字列検索を実現する方式も，いくつか提案されている．Bocek ら (Bocek, Hunt, and Stiller 2007) は，データベースに文字列を格納するときに，元の文字列に近い複数の隣接文字列を格納するアプローチ（隣接文字列生成）として，Fast Similarity Search (FastSS) を提案した．Wang ら (Wang, Xiao, Lin, and Zhang 2009) は，隣接文字列生成手法を改善するため，文字列を分割したり，接頭辞で枝刈りを行う方法を紹介した．Huynh ら (Huynh, Hon, Lam, and Sung 2006) は，圧縮された接尾辞配列上で類似文字列検索を行うアルゴリズムを提案した．Liu ら (Liu, Li, Feng, and Zhou 2008) は，文字列をトライに格納し，類似文字列検索を行う枠組みを提案した．これまでに紹介した研究は，編集距離を類似度関数として採用した場合に特化している．

Chaudhuri ら (Chaudhuri et al. 2006) は，編集距離とジャッカード係数に対する類似文字列検索に向けて，SSJoin 演算を提案した．このアルゴリズムは，検索クエリ文字列からシグニチャを作成し，シグニチャの特徴を含む全ての文字列を解候補として検索し，編集距離やジャッカード係数の制約を満たす文字列を選び出すものである．Chaudhuri らは，関係データベースの等結合（equi-join）を用いて SSJoin 演算を実装する方法を示した．本論文では，SSJoin 演算を関係データベース上で実装していないが，これは第 4 節の Signature システムと同等である．

Sarawagi ら (Sarawagi and Kirpal 2004) は，τ オーバーラップ問題を解くアルゴリズムとして，MergeOpt を提案した．このアルゴリズムは，転置リストを S と L という 2 つのグループに分け，S で解の候補生成を行い，L で解の検証を行う．提案手法と異なる点は，S で解の候補生成を行うときにヒープを用いる点，L で解の検証を行うときに，枝刈りを行わない点である．Li ら (Li et al. 2008) は，Sarawagi らの手法を改良し，SkipMerge と DivideSkip というアルゴリズムを提案した．SkipMerge アルゴリズムは，全ての転置リストの先頭から順に SID をヒープに挿入し，ヒープの先頭から同じ SID の要素を取り出したとき，取り出された個数が τ を超えたら，その SID を解とするものである．ただし，ヒープに転置リストから SID を挿入するときに，τ オーバーラップ問題の解となり得ない要素をスキップするメカニズムが組み込まれており，転置リスト中の全ての SID をヒープに挿入しなくても，τ オーバーラップ問題が解けるように工夫されている．DivideSkip アルゴリズムは，MergeOpt アルゴリズムと同様，転置リストを S と L という 2 つのグループに分け，SkipMerge アルゴリズムを S に適用して解の候補生成を行い，L で解の検証を行うものである．しかしながら，DivideSkip アルゴリズムでは解の枝刈り方法については，述べられていない．これらの手法と提案手法を解析的に比較するのは難

しいが，第 4 節では，SkipMerge と DivideSkip アルゴリズムによる類似文字列検索の性能を測定し，提案手法の方が高速に検索できることを実験的に示した．

続いて，提案手法と MergeOpt，SkipMerge，DivideSkip を空間計算量に関して比較する．ここに挙げた全てのアルゴリズムは，転置リスト上で二分探索を行うため，特殊な工夫をしない限り，転置リストの内容を主記憶に読み込む必要がある．最悪の場合を考えると，どのアルゴリズムも与えられたクエリ文字列に対して，最もサイズの大きい転置リストを主記憶に読み込む必要が生じる[16]．これに加え，各アルゴリズムとも解文字列の候補を主記憶上に保持しておく必要がある．MergeOpt，SkipMerge，DivideSkip アルゴリズムは，解候補をヒープに格納するアルゴリズムであり，ヒープに格納される解候補の数は，クエリに対する転置リストの数（すなわち，クエリの特徴集合 X の要素数 $|X|$）を超えない．これに対し，提案手法は，いったん解候補の列挙を行うため，おおよそ (転置リストの平均要素数) $\times$ ($|X| - \tau + 1$) 程度の解候補を主記憶に保持することになる．したがって，提案手法は MergeOpt，SkipMerge，DivideSkip アルゴリズムよりも空間計算量が大きくなる．表 6 には，提案手法の各データセットにおける解候補数（# 候補）が示されている．これによると，途中で保持した解候補数は数百〜数千程度のオーダーであり，提案手法の空間計算量は実用上は問題にならないと考えられる．

最後に，類似文字列検索に近いタスクとして，類似文字列照合との関係を説明する．このタスクでは，与えられた 2 つの文字列の表記が近いかどうかを精密に検出するため，文字列の類似度関数を改良したり (Winkler 1999; Cohen, Ravikumar, and Fienberg 2003)，機械学習で獲得するアプローチ (Bergsma and Kondrak 2007; Davis, Kulis, Jain, Sra, and Dhillon 2007; Tsuruoka, McNaught, Tsuji, and Ananiadou 2007; Aramaki, Imai, Miyo, and Ohe 2008) が取られる．これらの研究成果を用いると，2 つの文字列の類似性を高精度に判別できるが，判別する 2 つの文字列があらかじめ与えられることを前提としている．このような精細な類似度で類似文字列検索を行いたい場合は，適当な類似度関数に対して緩い閾値 α を用い，提案手法で類似文字列の候補を獲得してから，類似文字列照合を適用すればよい．

6　まとめ

本論文では，コサイン係数，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た類似文字列検索のための，新しいアルゴリズムを提案した．類似文字列検索を，τ オーバーラップ問題に帰着させ，その簡潔かつ高速なアルゴリズムとして CPMerge を提案した．このアルゴリズムは，τ オーバーラップ問題の解候補をできるだけコンパクトに保ち，解を効率よく求めるものである．英語の人名，日本語の単語，生命医学分野の固有表現を文字列データとして，

[16] 細かい議論になるが，SkipMerge と DivideSkip では複数の転置リスト上で並行して二分探索を行うため，特殊な工夫を施さない限り，複数の転置リストを同時に主記憶に読み込む必要が生じる．

類似文字列検索の性能を評価した．CPMerge アルゴリズムは非常にシンプルであるが，類似文字列検索の最近の手法である Locality Sensitive Hashing (LSH) (Andoni and Indyk 2008) や DivideSkip (Li et al. 2008) と比べ，高速かつ正確に文字列を検索できることを実証した．自然言語処理を実テキストに適用するときの基礎的な技術として，本研究の成果が活用されることを期待している．

CPMerge アルゴリズムが従来手法（例えば MergeSkip）に対して特に有利なのは，全ての転置リストを主記憶に読み込まなくても，類似文字列検索の解を求めることができる点である．表 6 に示した通り，CPMerge アルゴリズムはクエリに対して約 50%の転置リストを読み込むだけで，類似文字列検索の解を求めることができた（コサイン類似度で閾値が 0.7 の場合）．提案手法と従来手法は，アルゴリズム中で二分探索を用いるため，転置リスト上におけるランダムアクセスを，（暗黙的に）仮定している．したがって，読み込む転置リストの数を減らすことができるという提案手法の特徴は，転置リストを圧縮する際に有利であると考えられる．転置リストの圧縮に関する最近の研究成果 (Behm, Ji, Li, and Lu 2009; Yan, Ding, and Suel 2009) を参考に，圧縮された転置リストを用いた類似文字列検索を今後検討したいと考えている．

謝　辞

本研究は，岡崎が東京大学大学院情報学環，辻井が東京大学大学院情報学環，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に所属していた際に進められたものである．本研究の一部は，科学技術振興調整費・重要課題解決型研究等の推進「 日中・中日言語処理技術の開発研究」，文部科学省科学研究費補助金特別推進研究「高度言語理解のための意味・知識処理の基盤技術に関する研究」の助成を受けたものである．本論文に関して，大変有益かつ丁寧なコメントを頂いた査読者の方々に，感謝の意を表する．

付録: その他の類似度関数の条件式の導出

ダイス関数の場合

ダイス関数の定義は，

$$\mathrm{dice}(\mathrm{X},\mathrm{Y}) = \frac{2|\mathrm{X}\cap\mathrm{Y}|}{|\mathrm{X}|+|\mathrm{Y}|} \tag{8}$$

この関数の値が α 以上となる必要十分条件は,

$$\alpha \leq \frac{2|X \cap Y|}{|X| + |Y|} \tag{9}$$

$$\frac{1}{2}\alpha(|X| + |Y|) \leq |X \cap Y| \leq \max\{|X|, |Y|\} \tag{10}$$

$|X \cap Y|$ は整数であることに注意すると，必要十分条件が得られる．

$$\left\lceil \frac{1}{2}\alpha(|X| + |Y|) \right\rceil \leq |X \cap Y| \leq \max\{|X|, |Y|\} \tag{11}$$

式 10 において，$|X \cap Y|$ の項を削除すると,

$$\frac{1}{2}\alpha(|X| + |Y|) \leq \max\{|X|, |Y|\} \tag{12}$$

$|X| < |Y|$ のとき，$|Y|$ について解くと,

$$\frac{\alpha}{2 - \alpha}|X| \leq |Y| \tag{13}$$

$|X| \leq |Y|$ のとき，$|Y|$ について解くと,

$$|Y| \leq \frac{2 - \alpha}{\alpha}|X| \tag{14}$$

これらをまとめ，$|Y|$ が整数であることに注意すると，$|Y|$ の必要条件が得られる．

$$\left\lceil \frac{\alpha}{2 - \alpha}|X| \right\rceil \leq |Y| \leq \left\lfloor \frac{2 - \alpha}{\alpha}|X| \right\rfloor \tag{15}$$

ジャッカード関数の場合

ジャッカード関数の定義は,

$$\mathrm{jaccard(X, Y)} = \frac{|\mathrm{X} \cap \mathrm{Y}|}{|\mathrm{X} \cup \mathrm{Y}|} = \frac{|\mathrm{X} \cap \mathrm{Y}|}{|\mathrm{X}| + |\mathrm{Y}| - |\mathrm{X} \cap \mathrm{Y}|} \tag{16}$$

この関数の値が α 以上となる必要十分条件は,

$$\alpha \leq \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|} \tag{17}$$

$$\frac{\alpha}{1 + \alpha}(|X| + |Y|) \leq |X \cap Y| \leq \max\{|X|, |Y|\} \tag{18}$$

$|X \cap Y|$ は整数であることに注意すると，必要十分条件が得られる．

$$\left\lceil \frac{\alpha}{1 + \alpha}(|X| + |Y|) \right\rceil \leq |X \cap Y| \leq \max\{|X|, |Y|\} \tag{19}$$

式 18 において，$|X \cap Y|$ の項を削除すると，

$$\frac{\alpha}{1+\alpha}\left(|X|+|Y|\right) \leq \max\left\{|X|,|Y|\right\} \tag{20}$$

$|X| < |Y|$ のとき，$|Y|$ について解くと，

$$\alpha|X| \leq |Y| \tag{21}$$

$|X| \leq |Y|$ のとき，$|Y|$ について解くと，

$$|Y| \leq \frac{|X|}{\alpha} \tag{22}$$

これらをまとめ，$|Y|$ が整数であることに注意すると，$|Y|$ の必要条件が得られる．

$$\lceil \alpha|X| \rceil \leq |Y| \leq \left\lfloor \frac{|X|}{\alpha} \right\rfloor \tag{23}$$

オーバーラップ関数の場合

オーバーラップ関数の定義は，

$$\mathrm{overlap(X,Y)} = \frac{|\mathrm{X} \cap \mathrm{Y}|}{\min\{|\mathrm{X}|,|\mathrm{Y}|\}} \tag{24}$$

この関数の値が α 以上となる必要十分条件は，

$$\alpha \leq \frac{|X \cap Y|}{\min\{|X|,|Y|\}} \tag{25}$$

$$\alpha \min\{|X|,|Y|\} \leq |X \cap Y| \leq \max\{|X|,|Y|\} \tag{26}$$

$|X \cap Y|$ は整数であることに注意すると，必要十分条件が得られる．

$$\lceil \alpha \min\{|X|,|Y|\} \rceil \leq |X \cap Y| \leq \max\left\{|X|,|Y|\right\} \tag{27}$$

式 26 において，$|X \cap Y|$ の項を削除すると，

$$\alpha \min\{|X|,|Y|\} \leq \max\{|X|,|Y|\} \tag{28}$$

ここで，$\min\{|X|,|Y|\} \leq \max\{|X|,|Y|\}$，$0 \leq \alpha \leq 1$ であるから，式 28 は Y や α の選び方に依らず，常に成立する．従って，オーバーラップ関数を用いた類似文字列検索では，$|Y|$ に関する必要条件はない．

参考文献

Ahmad, F. and Kondrak, G. (2005). "Learning a spelling error model from search query logs." In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT-EMNLP 2005)*, pp. 955–962.

Andoni, A. and Indyk, P. (2008). "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions." *Communications of the ACM*, **51** (1), pp. 117–122.

Aramaki, E., Imai, T., Miyo, K., and Ohe, K. (2008). "Orthographic Disambiguation Incorporating Transliterated Probability." In *IJCNLP 2008: Proceedings of the Third International Joint Conference on Natural Language Processing*, pp. 48–55.

Arasu, A., Ganti, V., and Kaushik, R. (2006). "Efficient exact set-similarity joins." In *VLDB '06: Proceedings of the 32nd International Conference on Very Large Data Bases*, pp. 918–929.

Behm, A., Ji, S., Li, C., and Lu, J. (2009). "Space-Constrained Gram-Based Indexing for Efficient Approximate String Search." In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pp. 604–615.

Bergsma, S. and Kondrak, G. (2007). "Alignment-Based Discriminative String Similarity." In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL 2007)*, pp. 656–663.

Bocek, T., Hunt, E., and Stiller, B. (2007). "Fast Similarity Search in Large Dictionaries." Tech. rep. ifi-2007.02, Deportment of Informatics (IFI), University of Zurich.

Brill, E. and Moore, R. C. (2000). "An improved error model for noisy channel spelling correction." In *Proceedings of the 38th Annual Meeting on the Association for Computational Linguistics (ACL 2000)*, pp. 286–293.

Chaudhuri, S., Ganti, V., and Kaushik, R. (2006). "A Primitive Operator for Similarity Joins in Data Cleaning." In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering*, pp. 5–16.

Chen, Q., Li, M., and Zhou, M. (2007). "Improving Query Spelling Correction Using Web Search Results." In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007)*, pp. 181–189.

Cohen, W. W., Ravikumar, P., and Fienberg, S. E. (2003). "A comparison of string distance metrics for name-matching tasks." In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, pp. 73–78.

Davis, J. V., Kulis, B., Jain, P., Sra, S., and Dhillon, I. S. (2007). "Information-theoretic

metric learning." In *ICML '07: Proceedings of the 24th International Conference on Machine Learning*, pp. 209–216.

Gravano, L., Ipeirotis, P. G., Jagadish, H. V., Koudas, N., Muthukrishnan, S., and Srivastava, D. (2001). "Approximate String Joins in a Database (Almost) for Free." In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 491–500.

Huynh, T. N. D., Hon, W.-K., Lam, T.-W., and Sung, W.-K. (2006). "Approximate string matching using compressed suffix arrays." *Theoretical Computer Science*, **352** (1-3), pp. 240–249.

Jongejan, B. and Dalianis, H. (2009). "Automatic training of lemmatization rules that handle morphological changes in pre-, in- and suffixes alike." In *ACL-IJCNLP '09: Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1*, pp. 145–153.

Kim, M.-S., Whang, K.-Y., Lee, J.-G., and Lee, M.-J. (2005). "n-Gram/2L: a space and time efficient two-level n-gram inverted index structure." In *VLDB '05: Proceedings of the 31st International Conference on Very Large Data Bases*, pp. 325–336.

Lee, H., Ng, R. T., and Shim, K. (2007). "Extending q-grams to estimate selectivity of string matching with low edit distance." In *VLDB '07: Proceedings of the 33rd International Conference on Very Large Data Bases*, pp. 195–206.

Li, C., Lu, J., and Lu, Y. (2008). "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.

Li, C., Wang, B., and Yang, X. (2007). "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.

Li, M., Zhang, Y., Zhu, M., and Zhou, M. (2006). "Exploring distributional similarity based models for query spelling correction." In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (Coling-ACL 2006)*, pp. 1025–1032.

Liu, X., Li, G., Feng, J., and Zhou, L. (2008). "Effective Indices for Efficient Approximate String Search and Similarity Join." In *WAIM '08: Proceedings of the 2008 The Ninth International Conference on Web-Age Information Management*, pp. 127–134.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Okazaki, N., Tsuruoka, Y., Ananiadou, S., and Tsujii, J. (2008). "A discriminative candidate generator for string transformations." In *EMNLP '08: Proceedings of the Conference on*

Empirical Methods in Natural Language Processing, pp. 447–456.

Porter, M. F. (1980). "An algorithm for suffix stripping." *Program*, **14** (3), pp. 130–137.

Ravichandran, D., Pantel, P., and Hovy, E. (2005). "Randomized algorithms and NLP: using locality sensitive hash function for high speed noun clustering." In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pp. 622–629.

Sarawagi, S. and Kirpal, A. (2004). "Efficient set joins on similarity predicates." In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 743–754.

Tsuruoka, Y., McNaught, J., Tsuji, J., and Ananiadou, S. (2007). "Learning string similarity measures for gene/protein name dictionary look-up using logistic regression." *Bioinformatics*, **23** (29), pp. 2768–2774.

Wang, W., Xiao, C., Lin, X., and Zhang, C. (2009). "Efficient approximate entity extraction with edit distance constraints." In *SIGMOD '09: Proceedings of the 35th SIGMOD International Conference on Management of Data*, pp. 759–770.

Winkler, W. E. (1999). "The state of record linkage and current research problems." Tech. rep. R99/04, Statistics of Income Division, Internal Revenue Service Publication.

Xiao, C., Wang, W., and Lin, X. (2008a). "Ed-Join: an efficient algorithm for similarity joins with edit distance constraints." In *VLDB '08: Proceedings of the 34th International Conference on Very Large Data Bases*, pp. 933–944.

Xiao, C., Wang, W., Lin, X., and Yu, J. X. (2008b). "Efficient similarity joins for near duplicate detection." In *WWW '08: Proceeding of the 17th International Conference on World Wide Web*, pp. 131–140.

Yan, H., Ding, S., and Suel, T. (2009). "Inverted index compression and query processing with optimized document ordering." In *WWW '09: Proceedings of the 18th International Conference on World Wide Web*, pp. 401–410.

高橋克巳, 梅村恭司 (1995). "人名のかな表記のゆれに基づく近似文字列照合法." 情報処理学会論文誌, **36** (8), pp. 1906–1915.

獅々堀正幹, 津田和彦, 青江順一 (1994). "片仮名異表記の生成および統一手法." 電子情報通信学会論文誌. D-II, 情報・システム, II-情報処理, **77** (2), pp. 380–387.

略歴

岡崎 直観：2007 年東京大学大学院情報理工学系研究科・電子情報学専攻博士課程修了．同年，東京大学大学院情報理工学系研究科・特別研究員．2011 年より，東北大学大学院情報科学研究科准教授．自然言語処理，テキストマイニ

ングの研究に従事．情報理工学博士．情報処理学会，人工知能学会，ACL 各会員．

辻井 潤一：1971 年京都大学工学部，1973 年同修士課程修了．同大学助手・助教授を経て，1988 年英国 UMIST 教授，1995 年東京大学教授．マンチェスタ大学教授を兼任．2011 年，東京大学，マンチェスタ大学を退職．同年より，マイクロソフトリサーチアジア研究員（Principal Researcher），国立情報学研究所客員教授，マンチェスタ大学客員教授．TM，機械翻訳などの研究に従事．工博．ACL 元会長．