Renga Street: Top-k クエリ処理アルゴリズムによる高速類似文字列検索の実装

山城 颯太
LINEヤフー株式会社
soyamash@lycorp.co.jp

概要

類似文字列検索は，あらかじめ与えられた文字列集合のうちから，入力クエリ文字列に類似した文字列集合を返却するタスクであり，古くから研究され続けている [1, 2]．既存手法の多くは，類似性尺度として編集距離，あるいは文字 N-gram 素性（Q-gram）の cosine 係数や Jaccard 係数を使用しており，各素性ごとの重みを同一と見なしている．しかし，出現頻度がべき乗則に従う自然言語を対象とする場合は，BM25 [3] のように低頻度素性を重視するスコアリング手法を用いることで，文字列同士の比較回数削減と精度向上が見込める．本研究は類似文字列検索タスクを複数タームに基づく OR 検索として定式化し直し，Top-k クエリ処理手法の一つである MaxScore アルゴリズムを使用することで，高速かつ正確に Top-k 類似文字列が取得できることを確認した．

1 はじめに

類似文字列検索は自然言語を対象とした曖昧検索やスペル訂正，名寄せのみではなく，DNA 配列検索 [4]，類似音楽検索 [5] のような自然言語以外を対象とするアプリケーションにおいても使用される．そのため既存手法の多くは各素性の重みを均一に取り扱うが，自然言語のみを対象とする場合は，出現頻度に基づいて重み付けする IDF（Inverse Document Frequency）ベースのスコアリング手法を用いることで，より高精度な検索が実現できると予想される．しかし IDF ベースのスコアリング手法を用いる場合，Count Filtering，Length Filtering [1, 2] のような既存の類似文字列候補削減手法をそのまま適用することができず，文字列同士の比較回数が大幅に増加してしまう．本研究ではこれらの高速化手法の代わりに，Top-k クエリ処理を伴う複数ターム OR 検索としてタスクを定式化し直すことで，高速な類似文字列検索を実現する．

Top-k クエリ処理のうち，動的枝刈り [6] に着目する．動的枝刈りは上位 k 件に含まれないドキュメントのスコア評価を省略することで，Top-k OR 検索における転置インデックスの走査を高速化する．動的枝刈り手法としては MaxScore [7]，WAND [8]，Block-Max WAND [9] など，いくつか提案されているが，今回は MaxScore アルゴリズムを実装し，既存手法と比較する．本研究ではいずれの手法においても，素性として Q-gram を使用する．Q-gram とは文字 N-gram のことを指し，例えば「LINEヤフー」に含まれる $q = 2$ の Q-gram は [‘LI’, ‘IN’, ‘NE’, ‘Eヤ’, ‘ヤフ’, ‘フー’] となる．

2 関連研究

Okazaki et al. [10] は類似文字列検索タスクに対して，Count Filtering，Length Filtering 手法を効率的に適用する CPMerge アルゴリズムを提案し，様々な類似度スコアを用いる類似文字列検索ライブラリとして SimString[1)] を公開した．Count Filtering はクエリに対して指定された類似度以上となる文字列がクエリと共有すべき素性数の下限を求め，これを満たさない文字列候補を枝刈りする．Length Filtering はクエリに対して指定された類似度以上となる文字列が保持すべき素性数の上限と下限を求め，これを満たさない文字列候補を枝刈りする．しかし，そのアルゴリズムの性質上，SimString はユーザによる類似度の閾値の指定を必要とする．また，返却される類似文字列集合の要素間に順位付けは行われない．類似度の上位 k 件を求める場合は，返却された集合内各要素に対するクエリとの類似度を改めて計算し直し，順位付けする必要がある．今回実装する Top-k クエリ処理手法は，探索とスコアに基づく枝刈りを

1） https://www.chokkan.org/software/simstring/index.html.ja

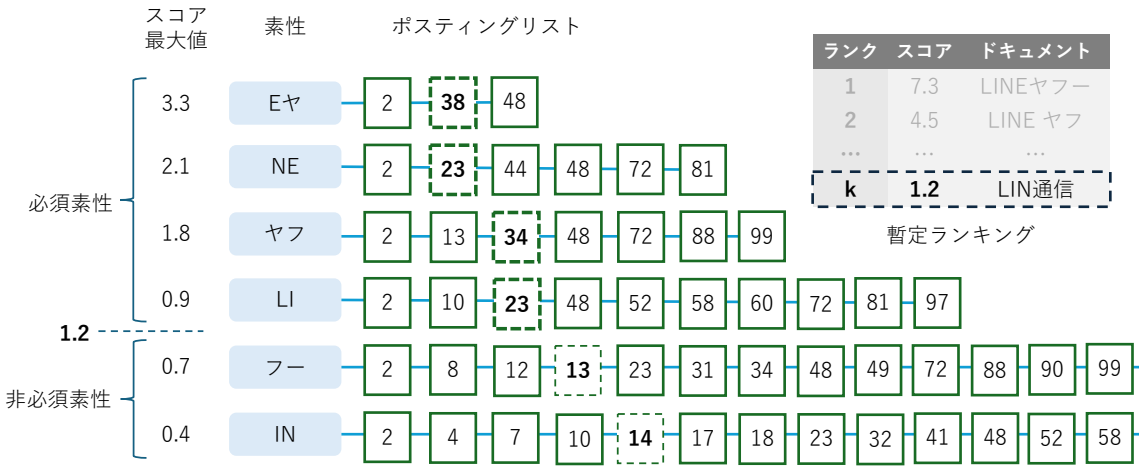

図 1　MaxScore アルゴリズムの実例

同時かつ動的に行うため，順位付けされた上位 k 件の類似文字列を高速に返却する．

3　実装手法

今回実装した MaxScore アルゴリズムを簡潔に説明する．より詳細な解説は [6] の 3.3 節を参照されたい．

先に，MaxScore アルゴリズムを使用しない通常の Top-k OR 検索について説明する．まず，クエリ中の各素性に対応するポスティングリストを取得する．ポスティングリストとはその素性を包含するドキュメントの ID のソート済み配列である．各ポスティングリストの先頭を指すカーソルを用意する．各カーソルが指すドキュメント ID のうち最も小さいドキュメント ID について，カーソル先ドキュメント ID が一致したポスティングリストからスコアを計算，合計し，カーソルを一つ先送りする．上記操作をすべてのドキュメント ID について行うことで，各ドキュメントのスコアが得られる．そのうち上位 k 件を返却する．

MaxScore アルゴリズムの実例を図 1 に示す．MaxScore アルゴリズムはスコア計算の対象となっているドキュメントのスコアが，現在暫定 k 位のドキュメントスコアに及ばないと判明した時点で探索を打ち切る．各素性について，その素性がクエリ中に出現した場合にドキュメントに加算されるスコア最大値をあらかじめ算出しておく．クエリ（図 1 の例においては「LINEヤフー」）が入力されたら，クエリ中の各素性に対応するポスティングリストを取得し，スコア最大値の降順に素性をソートする．また，現在時刻でスコア計算済みのドキュメントについてスコア順に並べた暫定ランキングを作成し，k 番目のドキュメントのスコアに基づいて，ソート済みクエリ素性リストの前半を必須素性，後半を非必須素性に分割する．このとき，非必須素性のポスティングリストのスコア最大値の総和（図 1 だと 0.7 + 0.4 =「1.1」）が，暫定 k 位のドキュメントのスコア（図 1 だと「1.2」）以下という条件を満たしつつ，非必須素性の個数が最大になるよう分割する．この分割によって，必須素性を一つも含まないドキュメントが非必須素性をすべて包含していたとしても，暫定 k 位のドキュメントよりスコアが小さくなることが保証される．従って，あるドキュメントが上位 k 件の類似文字列となる必要条件は，必須素性のうちのいずれか一つ以上を包含することである．つまり，必須素性のポスティングリストに出現しないドキュメント ID は，非必須素性のポスティングリスト上で探索する必要がなくスコア計算を省略できる．

従って実際のスコア計算手順としては，まず必須素性のカーソル（図 1 の太点線四角）が指すドキュメント ID のうち，最も小さいドキュメント ID（図 1 の「23」）について，カーソル先ドキュメント ID が一致したポスティングリストからスコアを計算し，カーソルを一つ先送りする．次に，非必須素性のカーソル（図 1 の細点線四角）を現在スコアを計算しているドキュメント ID 以上の ID が見つかるまで先送りし，出現していればスコアを足す．最終的なスコアが暫定 k 位のドキュメント（図 1 の「LIN通信」）より大きければ暫定ランキングを更新し，新たな暫定 k 位のドキュメントスコアに基づいて，必須素性と非必須素性の境界を更新する．暫定 k 位

のドキュメントスコアは次第に上昇していくので，探索が進むほどに必須素性の個数は減少していくことに注意されたい．以降上記手順を繰り返すことで，上位 k 件に含まれないドキュメントのスコア評価を省略しつつ，上位 k 件のスコアをすべて算出することができる．

このアルゴリズムは BM25 [3] などの IDF ベースのスコアを用いる場合のみ計算効率が良くなる．IDF はポスティングリストが短い素性ほどクエリ出現時に得られるスコアが大きくなるので，図 1 のように，非必須素性に対応するポスティングリストほど長くなり，かつ，スコア最大値が小さくなる．従って，MaxScore アルゴリズムを用いることにより，出現頻度が高い素性についてのスコア評価を省略することができ，全体として大幅な高速化が可能となる．

本研究では BM25 を使用する．これは低頻度素性の重視に加えて，より短いドキュメントを評価するスコアリング手法であり，素性 $q_1, ..., q_n$ を含むクエリ Q が与えられたとき，ドキュメント D に対する BM25 スコアは以下の式 (1) から得られる．

$$\mathrm{BM25}(D, Q) = \sum_{q_i \in Q} \mathrm{IDF}(q_i) \cdot \frac{\mathrm{TF}(q_i, D) \cdot (k+1)}{\mathrm{TF}(q_i, D) + k \cdot (1 - b + b \cdot \frac{|D|}{\mathrm{avg}|D|})} \tag{1}$$

ただし，$\mathrm{TF}(q_i, D)$ はドキュメント中の素性出現頻度，$|D|$ はドキュメント中の素性数．k，b は任意のパラメータでここでは $k = 1.2, b = 0.75$ とする．$\mathrm{IDF}(q_i)$ は以下の式 (2) で表される逆文書頻度で，N は全ドキュメント数，$n(q_i)$ は q_i を含むドキュメント数である．

$$\mathrm{IDF}(q_i) = \log\left(\frac{N}{n(q_i) + 1}\right) + 1 \tag{2}$$

4 実験

我々は 3 節の手法を C++で実装し，Renga Street と名付けた．各種タスクにおける Renga Street の性能を比較する．本研究では上記アルゴリズムを自前で実装したが，全文検索エンジンとして広く用いられる Elasticsearch[2] や Apache Solr[3] は Block-Max WAND [9] による Top-k クエリ処理機能を提供しており，手軽に利用できる．

2） https://www.elastic.co/elasticsearch
3） https://solr.apache.org

4.1 設定

類似文字列検索タスクの実験データとして日本語 Wikipedia[4] 記事本文中のハイパーリンクを抜き出し，5 回以上出現したリンク元表記・リンク先記事タイトルペアを利用した．[5] ランダムに選んだ 10,000 件のリンク元表記をクエリとし，リンク先タイトル集合（103,384 件）から正しいリンク先タイトルを引くことができたら正解とする．クエリの平均長は 9.6 文字，タイトル集合の平均長は 11.3 文字である．また，類似文検索タスクの実験データとして機械翻訳による言い換え文対データセットである JParaBank [11] を使用する．検証データの言い換え文 2,000 件をクエリとし，検証データの原文 2,000 件と訓練データの原文 100 万件を合わせた 1,002,000 件の文集合から正しい対応文を引くことができたら正解とする．クエリの平均長は 61.3 文字，文集合の平均長は 67.1 文字である．Renga Street の素性は表記に含まれる Q-gram（$q = 2$）とする．評価指標は MRR@k と Recall@k を用いる[6]．また，query/sec 列は著者の実行環境における 1 秒当たりに処理できたクエリ数であり，辞書の作成時間は含めない．

4.2 ベースライン

SimString をベースラインとして使用する．類似性尺度として Q-gram（$q = 2$）単位の cosine 係数を使用する．仕様上，クエリとともに尺度の閾値を入力する必要があるが，Top-k の候補を得るために設定すべき閾値は明らかでない．また，SimString の出力にはスコアが含まれないため，Top-k の順位付けを得るために改めて別のスコアリングを行う必要がある．今回の実験では，0.95 から 0.05 までの範囲を 0.1 刻み，あるいは 0.05 刻みで減少させて複数回検索を行い，指定された k 個以上の出力が得られた時点で検索を止める．[7] 出力された候補文字列に対

4） Wikipedia データは森羅プロジェクトが再配布しているバージョンを使用した．http://shinra-project.info/
5） Okazaki et al. [10] は検索対象文字列のうちの数文字をランダムに置換した人工クエリを評価用データとしている．今回我々が用意したデータは，人間が実際に記述した，より自然な表記ゆれのデータとなっている．
6） MRR@1 は Recall@1 と同じ値になるので省略している
7） 別案として，0.1 程度の緩い閾値を用いた検索を一度のみ行い，その中から上位 k 件を取得する方法が考えられる．この場合，検索回数は減る一方，類似文字列が大量に存在するようなクエリに対して取得できる候補数が肥大化し，却って編集距離・cosine 距離によるランキング部分の計算量が増大したため，複数回検索を採用している．

表 1　類似文字列検索タスク実験結果（Wikipedia ハイパーリンク）

	Top 1		Top 5			Top 10		
	Recall@1	query/sec	MRR@5	Recall@5	query/sec	MRR@10	Recall@10	query/sec
SimString(0.1) + 編集距離	51.3	6,349	47.6	60.8	2,526	47.0	64.7	1,872
SimString(0.05) + 編集距離	52.6	4,845	48.6	63.1	1,871	47.6	66.4	1,478
SimString(0.1) + cosine 係数	53.5	5,650	59.6	68.7	2,388	60.1	71.8	1,793
SimString(0.05) + cosine 係数	53.4	4,733	59.6	68.8	1,818	60.0	71.8	1,421
Renga Street	**58.3**	**16,502**	**63.3**	**70.5**	**10,142**	**63.7**	**73.4**	**9,390**

表 2　類似文検索タスク実験結果（JParaBank）

	Top 1		Top 5			Top 10		
	Recall@1	query/sec	MRR@5	Recall@5	query/sec	MRR@10	Recall@10	query/sec
SimString(0.1) + 編集距離	75.1	37.5	72.9	77.6	12.2	72.9	79.7	10.5
SimString(0.05) + 編集距離	75.6	25.5	73.4	78.7	8.6	73.4	80.6	8.4
SimString(0.1) + cosine 係数	74.4	37.9	76.5	81.2	12.7	76.8	83.7	11.1
SimString(0.05) + cosine 係数	74.7	25.7	76.8	81.7	8.8	76.9	83.6	7.9
Renga Street	**81.8**	**59.2**	**84.2**	**87.9**	**24.7**	**84.3**	**89.1**	**22.8**

して編集距離[8)]，あるいは Q-gram（$q = 2$）単位の cosine 係数を算出し，このスコアをもとにランキングした上位 k 件を最終的な出力と見なす．

4.3　結果

類似文字列検索の結果を表 1 に記す．本実験条件において，全体的に Renga Street がより正確な検索を実現できており，速度的にも SimString を用いた場合より数倍程度速いことがわかる．Renga Street は MRR@k スコアで SimString より高い数値を出しているのに加えて，Recall@10 においても依然として数 pt の差が残っていることから，後続のランキング処理の影響を除いた集合取得部分においても SimString より優れているのだと考えられる．SimString 同士を比較すると，後続のランキングには cosine 係数を使用したほうがスコアが高くなることがわかる．これは今回の Wikipedia ハイパーリンク元の表記とリンク先タイトルのペアとして，「1707 年連合法」，「連合法 (1707 年)」のようなフレーズ単位での交換例が含まれることから，語順を考慮しない cosine 係数のほうが適切だったと考えられる．後続のランキングに編集距離を用いた場合は，k の値を増やすと MRR@k の値が低下しており，編集距離が上手く順位付けできていないことを示している．また，編集距離を用いた場合の閾値の刻み幅は，より細かく変化させたほうがスコアが高くなることがわかる．これも編集距離が上手く順位付けできていないため，ランキング候補が増えるほど誤り率が増加しているのだと考えられる．Appendix A に出力例とともに定性評価を記す．

類似文検索の結果を表 2 に記す．本実験条件において，全体的に Renga Street がより正確な検索を実現できており，速度的にも SimString を用いた場合より数倍程度速いことがわかる．表 1 の類似文字列検索の結果と比較して，いずれの手法においても推論速度が低下している．これは検索対象の件数が増加しているのに加えて，1 サンプルあたりの素性数が多くなり比較にかかる時間が増加するためである．用いる素性を形態素に置き換えるなど工夫の余地が存在する．SimString 同士を比較すると，Recall@1 においてはランキングに編集距離を用いたほうがスコアが高くなる．これは JParaBank が比較的短い文で構成されており，言い換え文の語順が原文からほぼ変わらなかったため編集距離が上手く機能したものと思われる．ただし，文字列検索の場合同様，k の値を増やすと編集距離の MRR@k の値が低下し，cosine 係数を用いたほうがスコアが高くなっているので，一概にどちらの順位付けが適しているかは判断できない．一方，BM25 を用いた Renga Street は常にこれらの順位付け手法のスコアを上回っており，本実験条件において，よりタスクに適していることがわかる．

5　まとめ

我々は，自然言語を対象とした類似文字列検索タスクを Top-k クエリ処理を伴う複数ターム OR 検索として定式化し直すことで，既存ライブラリである SimString よりも高速・高精度な類似文字列検索が実現できることを確認した．

8）　編集距離の計算には Bit-Parallel アルゴリズム [12] を実装した

参考文献

[1] Yu Jiang, Guoliang Li, Jianhua Feng, and Wen-Syan Li. String Similarity Joins: An Experimental Evaluation. **Proc. VLDB Endow.**, Vol. 7, No. 8, pp. 625–636, April 2014.

[2] Minghe Yu, Guoliang Li, Dong Deng, and Jianhua Feng. String Similarity Search and Join: a Survey. **Frontiers of Computer Science**, Vol. 10, pp. 399–417, November 2015.

[3] Stephen Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In **Overview of the Third Text REtrieval Conference (TREC-3)**, pp. 109–126. Gaithersburg, MD: NIST, January 1995.

[4] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic Local Alignment Search Tool. **Journal of Molecular Biology**, Vol. 215, No. 3, pp. 403–410, 1990.

[5] Marcel Mongeau and David Sankoff. Comparison of Musical Sequences. **Computers and the Humanities**, Vol. 24, No. 3, pp. 161–175, 1990.

[6] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. Efficient Query Processing for Scalable Web Search. **Foundations and Trends in Information Retrieval**, Vol. 12, No. 4-5, pp. 319–500, 2018.

[7] Howard Turtle and James Flood. Query Evaluation: Strategies and Optimizations. **Information Processing and Management**, Vol. 31, No. 6, pp. 831–850, November 1995.

[8] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient Query Evaluation using a Two-level Retrieval Process. In **Proceedings of the Twelfth International Conference on Information and Knowledge Management**, CIKM '03, pp. 426–434, New York, NY, USA, 2003. Association for Computing Machinery.

[9] Shuai Ding and Torsten Suel. Faster Top-k Document Retrieval Using Block-Max Indexes. In **Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval**, SIGIR '11, pp. 993–1002, New York, NY, USA, 2011. Association for Computing Machinery.

[10] Naoaki Okazaki and Jun'ichi Tsujii. Simple and Efficient Algorithm for Approximate Dictionary Matching. In Chu-Ren Huang and Dan Jurafsky, editors, **Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)**, pp. 851–859, Beijing, China, August 2010. Coling 2010 Organizing Committee.

[11] 樽本空宙, 惟高日向, 梶原智之, 二宮崇. JParaBank：機械翻訳に基づく大規模な日本語言い換え文対の収集. 人工知能学会全国大会論文集, Vol. JSAI2023, pp. 4Xin113–4Xin113, 2023.

[12] Gene Myers. A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming. **J. ACM**, Vol. 46, No. 3, pp. 395–415, May 1999.

[13] Sosuke Nishikawa, Jun Hirako, Nobuhiro Kaji, Koki Watanabe, Hiroki Asano, Souta Yamashiro, and Shumpei Sano. Search Query Embeddings via User-behavior-driven Contrastive Learning. In Weizhu Chen, Yi Yang, Mohammad Kachuee, and Xue-Yong Fu, editors, **Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)**, pp. 138–147, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics.

表 3　類似文字列検索タスク（Wikipedia ハイパーリンク）の Top-5 出力例

クエリ文字列	SimString(0.05) + cosine 係数	Renga Street
カール 5 世	ダカール	**カール 5 世 (神聖ローマ皇帝)**
	カール 5 世 (神聖ローマ皇帝)	シャルル 5 世 (ロレーヌ公)
	カール大帝	ハーラル 5 世 (ノルウェー王)
	刑事犬カール 2	アレクサンデル 5 世 (対立教皇)
	カール・タウベ	カール 15 世 (スウェーデン王)
日本の平野	**平野**	日本国との平和条約
	日本の降伏	日本の降伏
	日本の首領	日本の首領
	日本の政党	日本の政党
	日本の仏教	日本の仏教
スイス連邦	スイス連邦鉄道	スイス連邦鉄道
	スイス	イギリス連邦
	スイス軍	連邦議会 (スイス)
	連邦	連邦大統領 (スイス)
	スイス傭兵	**スイス**

A　定性評価

表 3 に類似文字列検索タスク（Wikipedia ハイパーリンク）の Top-5 出力例を記す．各クエリに対する正解文字列を太字で表している．1 行目は「カール 5 世」をクエリとした各モデルの出力を示しており，SimString は正解文字列を 1 位として出力せず，Renga Street は正解文字列を 1 位と出力した事例として取り上げる．出力結果から，Renga Street は「ル 5」,「5 世」といった出現頻度の低い素性に着目することで,「(神聖ローマ皇帝)」という余分な文字列によるスコア低下を無視して正解することができている．2 行目は「日本の平野」をクエリとした各モデルの出力を示しており，SimString は正解文字列を 1 位として出力し，Renga Street は正解文字列を 1 位と出力しなかった事例として取り上げる．出力結果から，SimString は出現頻度の多い「平野」の一致を比較的高くスコア付けすることによって正解しており，Renga Street は「の平」,「本の」に対して高スコアを割り当ててしまったことで不正解となっている．ただし，このように高頻度素性のみで構成された正解タイトルは比較的少なかった．3 行目は「スイス連邦」をクエリとした各モデルの出力を示しており，両モデルとも正解文字列を 1 位として出力しなかった事例として取り上げる．出力結果から，SimString は「スイ」,「イス」に着目し，Renga Street は「ス連」に着目したことがわかる．

興味深い点として，各モデルの出力を見比べた際，Renga Street は出力集合に意味的なまとまりが見られることがわかる．Q-gram を素性とした BM25 は，ニューラルベースの埋め込み表現のように明示的に意味を学習してはいないが，コーパスにおける表層の頻度情報のみから，暗黙的に文字列の意味的カテゴリを取り扱うことができていると考えられる．表層のみに依存せず，より深い意味に着目した類似文字列検索が求められる場合は，検索クエリのように明示的な人間の行動を学習データとした文埋め込みモデル [13] を利用する手法が考えられる．このような連続値埋め込みベースの類似文字列検索との比較は今後の課題とする．