SoftMatcha 2: 1 兆語規模コーパスの超高速かつ柔らかい検索

米田優峻 [1,2]　鴨田豪 [3,4]　松下祐介 [5]　末永幸平 [2,5]　秋葉拓哉 [6,7]　和賀正樹 [2,5]　横井祥 [4,7,8]
[1] 東京大　[2]NII　[3] 総研大　[4] 国語研　[5] 京都大　[6]Sakana AI　[7] 東北大　[8] 理研
e869120@gmail.com　{go.kamoda, yokoi}@ninjal.ac.jp
{ymat, ksuenaga, mwaga}@fos.kuis.kyoto-u.ac.jp　takiba@sakana.ai

概要

超大規模コーパスを，超高速に，かつ意味や表記の揺れに柔らかく対応できる検索システム SoftMatcha 2 を提案・提供する．提案法は，言語モデルの学習コーパスの検索を目指す既存法の持つ特徴群，つまり，クエリの語順の保持，意味的類似性に基づいた置換，1 兆語規模コーパスに対する 0.1 秒単位の即時検索のすべてを満たし，さらにクエリへの単語の挿入や削除も考慮できる．特に，クエリに「似た」パターンの種類数の組合せ爆発という困難を，逐次的検索による枝刈りをおこなう新しいアルゴリズムで解決する．1.4 兆語のコーパス FineWeb-Edu [1] を用いた実験では，完全一致の場合も柔らかい検索の場合も，既存手法群（infini-gram [2], SoftMatcha [3]）と比べた大幅な高速化を確認した．さらに FineWeb-Edu [1][1)] および LLM-jp-corpus-v3 [4][2)] をウェブブラウザ上で高速検索できるオンラインツールを提供する．[3)]

1　はじめに

自然言語処理分野が作ってきた対話システムなどのアプリケーション群も，計算言語学・コーパス言語学などの諸分野が発見してきた言語に関する統計的・計量的な知見も，コーパスの存在に支えられてきた．とくに最近は，巨大コーパスを学習データとした言語モデルが我々の幅広い活動に欠かせざるものとなっており，モデルの意図せぬ挙動の原因追究 [5,6] や，ベンチマークデータの汚染度合いの推定 [7,8] など，巨大コーパスを高速に検索できるツールへの需要が一気に高まっている．

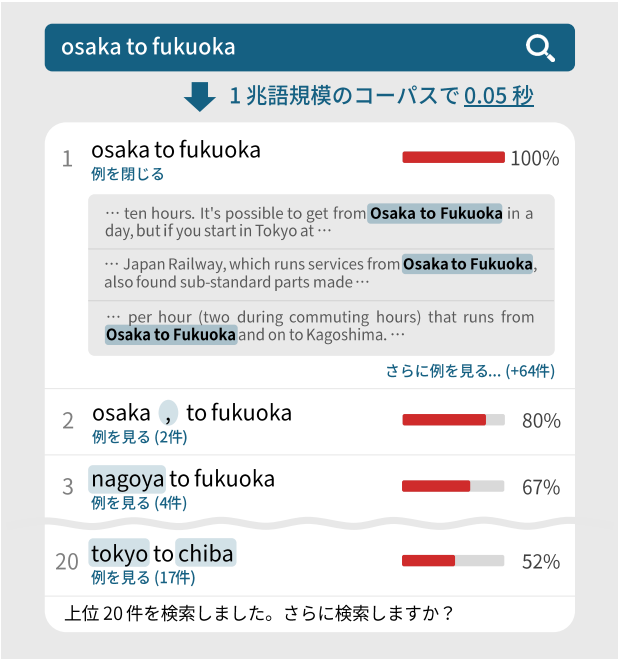

図 1　SoftMatcha 2 の検索例．パーセント表示は類似度．「例を見る」を押すと用例が表示される．

コーパスないし文書集合に対する検索自体は，情報検索などの領域で長く探求されてきたテーマである．これに対し，現代のコーパス検索のニーズの特殊性は次のような点にある：(1) 言語モデルの事前学習用にその規模がかつてないほど**巨大化**してきている（表 1）；(2) 言語モデルが非決定的・統計的に学習されるため，クエリに**類似**したテキストも検索したい；(3) 同時に言語モデルは次単語予測器（系列のモデル）であり，クエリの**語順**も考慮したい．実際，最近開発された各種のコーパス検索ツール [2,3,9–11] も，これらのニーズの一部を満たす（表 2）．本研究は，これらのニーズをすべて同時に実現するアルゴリズム・システムの開発を試みる．

実現したアプリケーションを図 1 に示す．提案アルゴリズムは，1 兆語規模の巨大コーパスに対して，意味的な類似性に加えてクエリと検索結果のトークン数のギャップにも対応しながら，高速に全

1）　ODC-By v1.0 License (https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu).
2）　コーパスの再配布は行わず，検索システムは AWS Tokyo Region で実行した (https://gitlab.llm-jp.nii.ac.jp/datasets/llm-jp-corpus-v3).
3）　ウェブツール: https://softmatcha.github.io/v2/.

図 2　検索クエリが “olympics gold medal” の場合における，提案手法の動作の例．

表 1　テキストコーパスの規模の拡大

	年	コーパス名	規模
英語	1967	Brown Corpus [12]	100 **万** 語
	2011	Gigaword [13]	40 **億** 語
	2024	FineWeb-Edu [1]	1.4 **兆** 語
日本語	2005	太陽コーパス [14]	1,400 **万** 字
	2011	現代日本語書き言葉均衡コーパス (BCCWJ) [15]	1 **億** 語
	2024	LLM-jp-corpus-v3 [4]	0.6 **兆** 語

表 2　先行研究との比較

手法	規模	意味的類似度	語順	全列挙	挿入削除
完全一致検索 (infini-gram [2] 等)	✓	✗	✓	✓	✗
密ベクトル検索	✓	✓	✗	✗	–
SoftMatcha [3]	✗	✓	✓	✓	✗
提案手法	✓	✓	✓	✓	✓

文検索を実行できる．実験では，言語モデルの事前学習データとして幅広く用いられる FineWeb-Edu [1]（約 1.4 兆語），日本語コーパス LLM-jp-corpus-v3 [4]（約 5,380 億語）に対し，柔らかい検索でも中央値約 0.05 秒の検索速度を実現した（図 3）．また，日本語・英語の両方のコーパスに対して実際に動く検索ウェブツール[3] を本稿に合わせて公開する．ぜひさまざまな検索を試していただきたい．

記法・問題設定　本研究で扱う検索システム（図 1 参照）のための記法を準備する．検索クエリである単語列を Q，出力件数 k を入力とし，対象コーパス中の単語列で Q との「類似度」が高い上位 k 件を出力する．類似度の定義は次セクションで検討する．

2　提案手法

アルゴリズム設計上のアイデア　柔らかい検索を提案した SoftMatcha [3] では転置インデックス自体を「柔らかく」しコーパスでの出現位置を直接列挙したが，我々は発想を転換し，柔らかくマッチする単語パターンの列挙を行う．これにより，従来手法では柔らかいパターンの列挙が困難だった，高速な接尾辞配列 [16] の活用が可能となる．ただし，素朴な全探索では候補数がパターン長に対して指数爆発を起こす．我々は，(1) 所与の件数の類似パターンを発見するまで類似度の閾値を段階的に徐々に緩和し，(2) n-gram の冪分布性を活用して分岐数を大幅に削減する，という工夫で探索空間の組合せ爆発を抑止する．

類似度の定義　パターン間の類似度の定義は検索の質に直結する．単語単位の意味的な類似性に対応するため，先行研究の SoftMatcha と同様に，単語埋め込み間のコサイン類似度に基づいてパターン間の類似度を定義する．[4] さらに，提案手法では，上記の類似度を単語の挿入・削除を許容するように拡張した．具体的には，挿入・削除ごとに，類似度を一定の割合減らす．[5] なお上記の類似度は，単語長に関して単調である．

提案アルゴリズム　アルゴリズムを詳しく説明する．クエリの類似度が閾値 α 以上のトークン列でコーパス中に出現するものを列挙し，列挙された件数が要求件数に満たない場合は α の値を下げて再度列挙を行うという手続きを繰り返す．以下，ある α について，提案アルゴリズムをクエリ “olympics gold medal” に対して動作させる例を考える（図 2）．

4)　SoftMatcha では単語列間の類似度として各単語間のコサイン類似度の最小値を採用したが，これには類似度が最低以外の単語の類似度を無視する問題があった．この問題を回避するため，提案手法では最小値以外の値によっても減る「滑らかな最小値」を用いる．詳細は付録 A を参照されたい．

5)　この際，単語ベクトルのノルムを用いて，the や of のように情報量の少ない単語ほど挿入・削除のコストを小さくする [17, 18].

表 3 提案手法の，英語・日本語コーパス全体における検索結果の例．カッコ内はヒット数，右端の列は既存手法 SoftMatcha でヒットするかどうかを示す．*1 は SoftMatcha，*2 は infini-gram でヒットするかを示す．

	検索クエリ：olympics gold medalist（英語）				検索クエリ：東京から大阪までの 2 時間半（日本語）			
順位	パターン	類似度	*1	*2	パターン	類似度	*1	*2
1	olympics gold medalist (838)	100.0%	✓	✓	東京から大阪までの 2 時間半 (11)	100.0%	✓	✓
2	olympics gold **medallist** (456)	89.1%	✓	✗	東京から大阪まで~~の~~ 2 時間半 (52)	87.5%	✗	✗
3	**olympic** gold medalist (95,816)	86.4%	✓	✗	東京から大阪まで**は** 2 時間半 (1)	77.9%	✓	✗
4	olympics **silver** medalist (500)	84.8%	✓	✗	東京から大阪まで~~の~~ **6** 時間半 (11)	75.8%	✗	✗
5	**olympic** gold **medallist** (22,506)	82.0%	✓	✗	東京から大阪まで**は 2** 時間半 (4)	64.4%	✓	✗
⋮								
20	olympics **,** **silver** medalist (94)	66.8%	✗	✗	東京から**仙台**まで**が 13** 時間半 (1)	52.3%	✓	✗

Step 1: クエリの最初のトークン “olympics” に類似するトークン列で類似度が α 以上であるものを全列挙する．[6] このステップはコーパスを参照せず，辞書データのみで実行する．列挙されたトークン列の集合を L_{1a} とする．その後，L_{1a} の各元がコーパス中に出現するか否かを，接尾辞配列を用いた完全一致検索で判定する．[7] L_{1a} の元のうちコーパス中に出現するトークン列の集合を L_{1b} とする．

Step 2: L_{1b} の各元（w とする）に対し，w を接頭辞として持ち “olympics gold” との類似度が α 以上となるトークン列 $w \cdot w'$ を全列挙する．[8] 列挙されたパターンを L_{2a} とする．以前のステップと同様に，L_{2a} の元のうちコーパス中に出現するもののみを残す．

Step 3 以降: 前ステップまでに列挙されたトークン列を接頭辞として持つパターンを，クエリにおいて着目する接頭辞を逐次的に拡張しつつ列挙する．

3 実験・デモ

コーパス 各言語の最大規模のコーパスを利用．[9]

- 英語：`FineWeb-Edu`（約 1.4 兆トークン）
- 日本語：`LLM-jp-corpus-v3`（約 5,380 億トークン）

検索クエリ Gemini 3 Pro を用い，日常語から専門用語まで幅広い内容のクエリを日英各 100 件生成．[10]

計算環境 AWS の `i8g.48xlarge` インスタンス（192 vCPU，RAM 1536GB，NVMe SSD 45TB）を利用．

単語ベクトル 学習済みモデルを [18] で後処理．[11]

- 英語：`glove-wiki-gigaword-300` [19]
- 日本語：`facebook/fasttext-ja-vectors` [20]

ベースライン手法 柔らかい検索と完全一致検索の想定される最速の先行研究をベースラインに設定．

- 柔らかい検索：単語埋め込みによる緩和を用いた直接的な先行研究である，SoftMatcha [3]
- 完全一致：接尾辞配列を用いた最新の検索ツールである，infini-gram [2] [12]

検索結果の出力 以下のパラメータによる．[13]

- 上位 $k = 20$ タイプの検索結果を表示．
- 類似度スコア 0.45 以上のパターンを表示．

3.1 検索結果

“olympics gold medalist”（英語版）および “東京から大阪までの 2 時間半”（日本語版）と検索した場合の結果を表 3 に示す．前者では，“olympics silver medalist” など類似の単語列がヒットしており，細かい意味の違いや表記揺れなどを捉えることができている．後者では，地名や時間を少し変えた単語列だけでなく，“東京から大阪まで 2 時間半” などトークン数が異なるパターンもヒットした．これらは既存の SoftMatcha では対象外であったため，*1 列に示す通り，検索の幅が大きく広がった．

3.2 実行時間

様々な規模の英語・日本語コーパスに対する，提案手法及びベースライン手法の検索時間の中央値および 95 パーセンタイル値を図 3, 4 に示す．図の一

6） 現在の実装においては，クエリの先頭トークンから探索を始めるか，クエリのうち最も出現頻度の低い単語から探索を始めるかを適応的に選択している．

7） 通常の接尾辞配列では，完全一致検索の際ディスクのランダムアクセスが $O(\log N)$ 回（N：コーパスサイズ）となり低速である．そこで，接尾辞配列全体を分割し，各ページ先頭の接尾辞のみからなる配列を主記憶に保持することで，ディスクへのアクセスが一回の完全一致探索につき 1 回で済むようにした．詳細は付録 B を参照されたい．

8） なお，この枝刈りの正しさは，前述した類似度の単語長に関する単調性により保証される．

9） 後述の単語ベクトルに対応してトークナイズした．

10） 英語は 2–10，日本語は 2–11 トークンの範囲で生成．

11） 英語は上位 40 万語彙，日本語は上位約 52 万語彙を使用．

12） 後続の infini-gram mini [9] よりも高速．

13） ベースラインの実験も同様．SoftMatcha はしきい値 α 以上のものをすべて出力する仕様のため，あらかじめ上位 k 番目の類似度 α を実行時間外で計算した後，その α の値を用いて検索を行うことで条件を揃える．なお検索範囲から一旦外れる類似度 0.45 以上かつ $k = 20$ 以上のクエリは，英語で 10%，日本語で 12%程度であった．

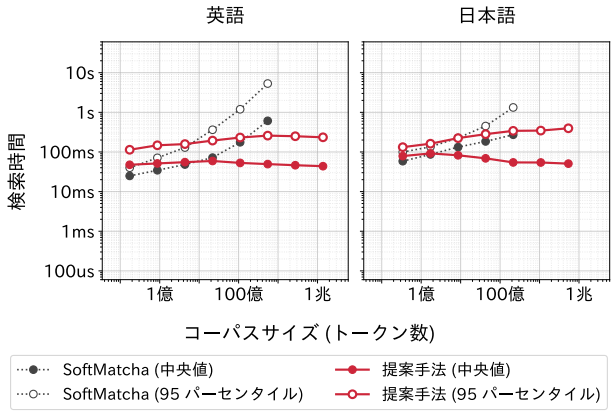

図 3　柔らかい検索のコーパスサイズに対する検索時間.

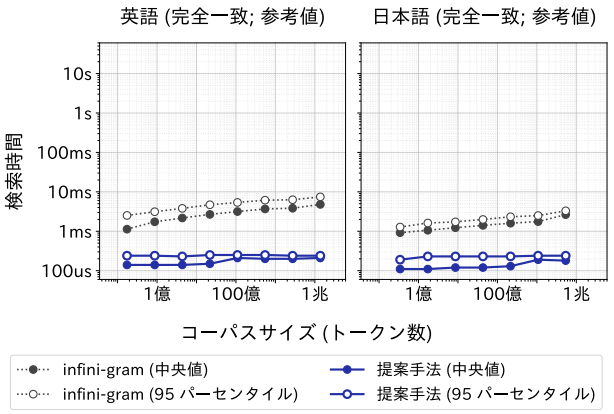

図 4　完全一致検索のコーパスサイズに対する検索時間.

番右の点はコーパス全体に，それより左の点はその部分サンプリングに対応する．

柔らかい検索との比較　提案手法は約 1.4 兆トークンの英語コーパスに対し中央値 43.75 ミリ秒で，約 5,380 億トークンの日本語コーパスに対し中央値 50.76 ミリ秒での全列挙が成立している．いずれの場合もコーパス検索ツールとして十分実用的な 0.1 秒単位の列挙が実現している．さらに提案手法は，たとえば株価の記事など定型的な文書の検索に有用な長いクエリに対しても，同英語コーパスに対し，トークン長 8, 9, 10 のクエリを中央値 96.49 ミリ秒（クエリ数 20），213.84 ミリ秒（クエリ数 20），271.60 ミリ秒（クエリ数 12），いずれも 0.3 秒未満で処理可能であった．一方で SoftMatcha は 5 億トークン程度までは提案手法と同程度の時間で検索可能であったが，それ以上大きなコーパスでは提案手法に劣る．特に，SoftMatcha は 10 億トークン以上のコーパスに対しては実行時間がほぼ線形で増加してしまう一方，提案手法は増加がほとんど見られず，日本語の中央値ではむしろ減少している．[14] また，SoftMatcha はメモリ不足のエラーにより 600 億トークン超のコーパスを扱えなかった．例えば 1,000 億トークンのコーパスに対して 2TB 以上のメモリを要し，巨大コーパスを現実的に扱えない．[15]

柔らかくない，完全一致検索との比較　参考のため，提案手法の一部である完全一致検索アルゴリズムと，接尾辞配列を活用した infini-gram を比較した（図 4）．ここでは意味的な緩和等は実現していないことに注意．提案手法はコーパスサイズによらず 0.21 ミリ秒程度で動作し，infini-gram に対して約 20 倍高速であった．高速かつ柔らかい検索の実現のために提案した完全一致検索アルゴリズムが，それ単体で有用であることを示すものである．

3.3　デモ

本研究では，実際に数千億語規模のコーパスに対する高速な用例検索のデモを **SoftMatcha 2** として https://softmatcha.github.io/v2/ で一般公開している．ぜひ試していただきたい．

4　おわりに

本研究では，1 兆トークン超のコーパスに対しても 0.1 秒単位で超高速に動作し，検索クエリの語順を保持し，さらに単語の意味的な類似性に基づく置換・挿入・削除に対応した検索システム SoftMatcha 2 を提案した．今後，自然言語処理や機械学習分野タスクのサブルーチンとしての活用，各言語の大規模コーパスのホスト，さらにアルゴリズムのさらなるスケールアップを検討したい．

14)　提案手法の実行時間が減少する理由として，コーパス中のトークン数が増えることによってより大きな類似度の閾値 α でも k 件のヒットを得られるようになり，列挙する単語パターンの個数が減少することが考えられる．

15)　SoftMatcha はインデックスをメモリ上に展開せず，SSD など二次記憶から読み込む実行モードも有する．この場合の実行時間も概ね同様の傾向を示したが，データへのアクセス時間の増加の影響で実行時間が 10 倍程度に増加した．

謝辞

本研究は，JSPS 科研費 JP24KJ0133, JP23K24910; JST CREST JPMJCR2012; JST 創発 JPMJFR2331; JST さきがけ JPMJPR22CA; JST BOOST JPMJBY24H8, JPMJBS2412; 京都大学白眉プロジェクトの支援を受けたものです．

出口祥之氏（NTT）から，SoftMatcha [3] の実装の詳細についてご助言をいただきました．感謝します．

参考文献

[1] Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. FineWeb-Edu: the Finest Collection of Educational Content, 2024.

[2] Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infini-gram: Scaling Unbounded n-gram Language Models to a Trillion Tokens. In **First Conference on Language Modeling**, 26 August 2024.

[3] Hiroyuki Deguchi, Go Kamoda, Yusuke Matsushita, Chihiro Taguchi, Kohei Suenaga, Masaki Waga, and Sho Yokoi. SoftMatcha: A Soft and Fast Pattern Matcher for Billion-Scale Corpus Searches. In **The Thirteenth International Conference on Learning Representations**, 4 March 2025.

[4] LLM-jp Corpus v3. Dataset, GitLab, 2024.

[5] Yiyou Sun, Yu Gai, Lijie Chen, Abhilasha Ravichander, Yejin Choi, Nouha Dziri, and Dawn Song. Why and How LLMs Hallucinate: Connecting the Dots with Subsequence Associations. In **The Thirty-ninth Annual Conference on Neural Information Processing Systems**, 29 October 2025.

[6] Ekin Akyurek, Tolga Bolukbasi, Frederick Liu, Binbin Xiong, Ian Tenney, Jacob Andreas, and Kelvin Guu. Towards tracing knowledge in language models back to the training data. In **Findings of the Association for Computational Linguistics: EMNLP 2022**, pp. 2429–2446. Association for Computational Linguistics, December 2022.

[7] Chunyuan Deng, Yilun Zhao, Xiangru Tang, Mark Gerstein, and Arman Cohan. Benchmark Probing: Investigating Data Leakage in Large Language Models. In **NeurIPS 2023 Workshop on Backdoors in Deep Learning - The Good, the Bad, and the Ugly**, 28 October 2023.

[8] Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. Benchmarking benchmark leakage in Large Language Models. **arXiv [cs.CL]**, 29 April 2024.

[9] Hao Xu, Jiacheng Liu, Yejin Choi, Noah A Smith, and Hannaneh Hajishirzi. Infini-gram mini: Exact n-gram Search at the Internet Scale with FM-Index. In **Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing**, pp. 24955–24980. Association for Computational Linguistics, November 2025.

[10] Adam Kilgarriff, Vít Baisa, Jan Bušta, Miloš Jakubíček, Vojtěch Kovář, Jan Michelfeit, Pavel Rychlý, and Vít Suchomel. The Sketch Engine: ten years on. **Lexicography**, Vol. 1, No. 1, pp. 7–36, July 2014.

[11] 中納言. https://chunagon.ninjal.ac.jp/.

[12] Henry Kučera and W Nelson Francis. **Computational Analysis of Present-Day American English**. Brown University Press, 1967.

[13] Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English Gigaword Fifth Edition. Web Download, 2011.

[14] 国立国語研究所. 太陽コーパス: 雑誌太陽日本語データベース. 2005.

[15] 前川喜久雄, 山崎誠（編）. 書き言葉コーパス-設計と構築-, 講座日本語コーパス, 第 2 巻. 朝倉書店, 2014.

[16] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In **Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms**, SODA '90, p. 319–327, USA, 1990. Society for Industrial and Applied Mathematics.

[17] Momose Oyama, Sho Yokoi, and Hidetoshi Shimodaira. Norm of word embedding encodes information gain. In **Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing**, pp. 2108–2130. Association for Computational Linguistics, December 2023.

[18] Sho Yokoi, Han Bao, Hiroto Kurita, and Hidetoshi Shimodaira. Zipfian Whitening. In **The Thirty-eighth Annual Conference on Neural Information Processing Systems**, 6 November 2024.

[19] Jeffrey Pennington, Richard Socher, and Christopher D Manning. GloVe: Global Vectors for Word Representation. In **Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)**, pp. 1532–1543, October 2014.

[20] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning Word Vectors for 157 Languages. In Nicoletta Calzolari, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Koiti Hasida, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, and others, editors, **Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)**. European Language Resources Association (ELRA), May 2018.

[21] Ranjan Sinha, Simon Puglisi, Alistair Moffat, and Andrew Turpin. Improving suffix array locality for fast pattern matching on disk. In **Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data**, SIGMOD '08, p. 661–672, New York, NY, USA, 2008. Association for Computing Machinery.

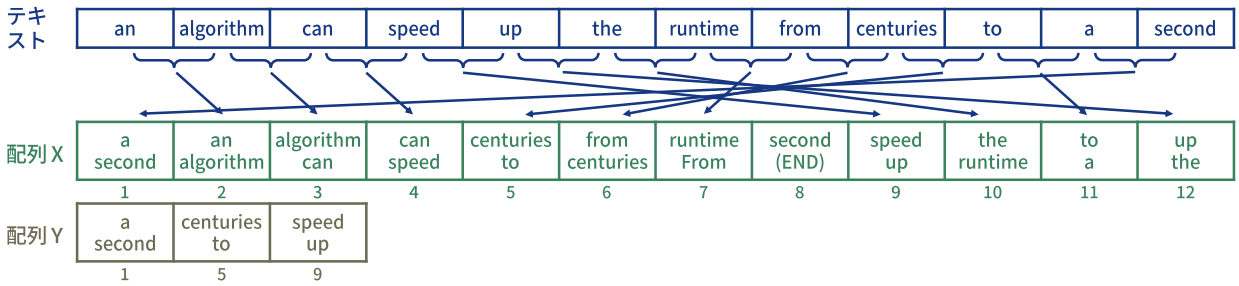

図 5　完全一致検索の高速化における，$K = 2$ および $B = 4$ の場合の例．

A　類似度の計算について

トークン数 m の検索クエリ $(p_1, \ldots, p_m)$ とパターン $(x_1, \ldots, x_m)$ の類似度は，挿入・削除を考慮しない場合，次式で定義される．ただし，$c_i \le 1$ を p_i と x_i のコサイン類似度とし，α を適当な定数とする．

$$1 - \log_\alpha \left(\sum_i (\alpha^{1-c_i} - 1) + 1 \right) \tag{1}$$

ここで，$\alpha = \infty$ の場合は式 1 は $\min(c_1, \ldots, c_m)$ となるが，本実験では $\alpha = 10^4$ を用いた．これは「滑らかな最小値 (smooth minimum)」の一種であり，最小値以外の値の影響でもわずかに減少する．

また，挿入・削除のコストは，Zipfian 白色化後のノルムの 2 乗を使用した．具体的には，挿入・削除する単語のノルムの 2 乗を v として，類似度に

$$\exp\left(-\frac{v}{\beta m}\right) \tag{2}$$

を掛ける．ただし m は**計算のどの段階においても**，実際の検索クエリのトークン数とする．定数 β は，トークン数が $m = 5$ の場合にノルムが下位 40 番目の単語を挿入した際の倍率が $1/e$ となるよう設定した（英語版で $\beta \approx 27.47$，日本語版で $\beta \approx 16.22$）．ここでノルムを使用した理由は，Zipfian 白色化後のノルムが情報量と強く相関する [17, 18] からである．

例　たとえば，gold と bronze のコサイン類似度が 0.63，medal と medals のコサイン類似度が 0.85 のとき，“gold medal” と “bronze medals” 間の類似度は約 0.62 であり，medal と medals 間のコサイン類似度が 1.0 未満であることが反映されている．また，Q = “part of cake” と X = “part of the chocolate” の類似度を考える．この場合，Q に the を挿入して “part of the cake” としてから各文字をマッチさせるのが一番もっともらしい．“the” の挿入コストが 0.82 倍，挿入後の類似度が 0.70 の場合，Q と X 間の類似度は $0.82 \times 0.70 \approx 0.57$ となる．

B　完全一致検索の高速化

接尾辞配列を用いた完全一致検索は，通常 $O(\log N)$ 回（N はコーパスのトークン数）のディスクアクセスが必要となるが，提案手法では検索クエリが短いことを使って，高速化を図った．ここで検索クエリのトークン数上限を K とする（実験では $K = 11$ とした）．

まず，接尾辞配列の代わりに，コーパスに出現する長さ K のトークン列をソートした配列 X を作成する（例は図 5 を参照）．ここで，配列 X 上の二分探索ができれば，コーパス内の完全一致の個数を求めることができる．たとえば，図 5 で “can speed” は前から 4 番目の位置であると，高速に計算できれば良い．しかし，配列 X だけでは依然として，$O(\log N)$ 回のディスクアクセスを要する．

そこで，ソート列を B 件毎[16] に区切った配列 Y を作成する [21]．配列 Y はメモリに収まるので，検索の際はまず配列 Y を使って大まかな二分探索を行い，次に配列 X を使って厳密な二分探索を行う，という二段階方式を取る．たとえば，図 5 で “can speed” の位置を検索する際，まず配列 Y で「前から 1–4 番目の間である」ことを特定し，次に配列 X で 4 番目であることを特定する．すると，二段階目の探索の際に読み込むべき配列 X の範囲は B エントリに収まる．OS が数 KB 単位でページを読み込むことを考慮すると，ディスクアクセスが事実上 1 回となり，大幅な高速化が期待される．

なお，配列 X のサイズは，語彙数を V として $N\lceil K \log_2 V/8 \rceil$ バイトとなり，1 兆語規模では巨大となる．しかし，引用等により，同じ単語列が数多く出現するため，ランレングス圧縮で大幅に圧縮される．たとえば，`FineWeb-Edu` (1.4 兆トークン) では，圧縮率約 86.3% となった．

16)　実験では $B = 128$ または 256 とした．