自然言語処理 121－12
情報学基礎 47－4
（1997. 9. 11）

Suffix Array を用いたフルテキスト類似用例検索

山下 達雄，松本 裕治
奈良先端科学技術大学院大学 情報科学研究科
{tatuo-y,matsu}@is.aist-nara.ac.jp

類似検索の手法として DP マッチングアルゴリズムやトライを用いた Error-tolerant Recognition アルゴリズムが知られている。DP マッチングアルゴリズムは検索対象データの大きさに比例した計算時間がかかるのに対し、トライを用いた Error-tolerant Recognition アルゴリズムは検索対象データの大きさに依存せず高速に類似検索が行える。しかし、トライにはデータ領域の非効率性の問題が、Error-tolerant Recognition アルゴリズムにはフルテキスト検索には向かないという問題点がある。本稿では、これらの問題を解決するための方法について述べる。まず Error-tolerant Recognition アルゴリズムをフルテキスト検索に拡張する。次にトライのデータ領域の問題を Suffix Array というデータ構造を用いた疑似的なトライを実現することにより解決する。これらにより、大規模なテキストデータに対するフルテキスト類似用例検索システムを実現した。

[キーワード] 全文検索，類似用例検索，トライ，Suffix Array，Error-tolerant Recognition

Full Text Approximate String Search using Suffix Arrays

YAMASITA Tatuo, MATSUMOTO Yuji
Graduate School of Information Science, Nara Institute of Science and Technology

The error-tolerant recognition algorithm using TRIE and the dynamic programming algorithm are known as methods of the approximate matching. Computational complexity of the dynamic programming algorithm is in proportion to the text size, while that of the algorithm using TRIE is independent of the text size. However, TRIE has a problem that it requires a huge amount of data space, and the error-tolerant recognition algorithm has a problem that it does not take account of full text search. In this paper, we give a solution to these problems of the error-tolerant recognition algorithm using TRIE. First, we extend the error-tolerant recognition algorithm and apply it to the full text search. Secondly, to solve the data space problem, we employ a data structure called Suffix Arrays to achieve a large scale full text approximate string search system.

[keyword] full text search, approximate string matching, trie, suffix arrays, error-tolerant recognition

1 はじめに

曖昧性を許す検索、つまり、入力データ (検索キーワード) と検索対象データの間の類似性を用いた検索は様々な分野において有用な技術である。例えば、用例に基づく機械翻訳 [8][10] や翻訳支援 [7][9] における類似用例検索、校正システムや OCR 読み取りにおけるスペル誤りの訂正 [12]、新聞などの文書データに対する類似検索を用いた情報検索 [13] など、自然言語処理の分野において用途は広い。ま

た、ゲノムサイエンスの分野においても、塩基・アミノ酸配列からの特定の部位の検索[1]やアライメントなどで用いられている [11][14]。

本研究では、このような類似検索のうち、フルテキスト類似用例検索の高速化の手法について論じる。

本稿の構成について述べる。第2章では、本研究で扱うフルテキスト類似検索のモデルについて説明する。フルテキスト類似検索と辞書類似検索の違いについても明確にする。第3章では、これまでに研究されてきた類似検索の手法について説明する。一つはDPマッチングによる方法で、スペルチェッカーや塩基配列・アミノ酸配列の類似検索によく用いられている。もう一つはトライによる方法で、DPマッチングに比べ計算量において優れている。第4章では、トライによる方法の問題点であるデータ領域の非効率性を解消するため、Suffix Arrayというデータ構造を用いて疑似トライを実現するという方法を提案する。

2 フルテキスト類似検索のモデル

本研究での対象となるフルテキスト類似検索のモデルを以下のように定義する。

- 検索キーワードに対し置換/挿入/削除の3つの基本操作(図1)が用意されている。各基本操作にはあらかじめペナルティが設定できる(図2)。
- 3つの基本操作を用いて、ペナルティの合計が最小になるように、検索キーワードと検索対象テキストの部分文字列を並べる。
- ペナルティの合計がある値(限度)以下のものだけを検索結果として出力する。

例えば、図2では、ギャップ(挿入/削除)ペナルティは2、B→CまたはC→Bの置換ペナルティは2、同じ文字同士の置換ペナルティは0、他の置換ペナルティは1となっている。すると、ペナルティの合計は、図1の置換の例では1、挿入/削除の例では2となる。もしペナルティの合計の限度が1とするならば、置換の例(ABA)だけが検索キーワードABCの類似検索結果として出力される。

ここでフルテキスト類似検索と辞書類似検索の違いを明確にしておく。

[1]ホモロジー検索と呼ばれている。

テキスト BABAC　検索キーワード ABC

- 検索キーワードの文字を置換

	A	B	C	
B	A	B	A	C

検索キーワードの3文字目のCをAとみなす。

- 検索キーワードに新たな文字を挿入

	A	B	−	C
B	A	B	A	C

検索キーワードの2文字目と3文字目の間にAを挿入する。

- 検索キーワードからある文字を削除

			A	B	C
B	A	B	A	−	C

検索キーワードの2文字目のBを無いものとする。

図 1: 基本操作

ギャップペナルティ	2			
置換ペナルティ	A	B	C	
	0	1	1	A
		0	2	B
			0	C

図 2: ペナルティの例

フルテキスト類似検索 検索対象テキストの中から検索キーワードと類似した部分文字列を探す。

辞書類似検索 すべての検索対象の文字列(見出し語)と検索キーワードを両方が同じ長さになるように並べてみて、ある類似性を充たす文字列を探す。

例えば図1の例では、両方の文字列の長さをそろえるために下図のように検索キーワードの両側にギャップを入れる必要がある。

−	A	B	C	−
B	A	B	A	C

検索対象のデータ量が同じ場合、当然フルテキスト検索の方が検索量が多くなる。用例ベース機械翻訳などの従来の類似用例の検索は辞書類似検索である [8]。塩基・アミノ酸配列のホモロジー検索はフルテキスト類似検索である

3 類似検索の手法

これまで研究されてきた類似検索の手法について説明する。

3.1 DP マッチング

n を検索対象テキストの長さ、m を検索キーワードの長さとする。テキスト $t_1, t_2, \cdots, t_n$ から検索キーワード $a_1, a_2, \cdots, a_m$ を類似検索することを考える。これまでに $a_1, a_2, \cdots, a_{i-1} (i \leq m)$ と $t_1, t_2, \cdots, t_{j-1} (j \leq n)$ の比較が終わっていたとして、そこまでのペナルティの合計が $P_{i-1,j-1}$ で表されているとする。この先比較を続ける際、可能性として、一致、不一致を問わず a_i と t_j を並べるか (置換)、挿入/削除を行うかしかない。従って、ペナルティの合計 $P_{i,j}$ を求めるには、この中でペナルティ最小の操作を選べば良い。これを式で表すと以下のようになる。s は置換ペナルティ、g はギャップ (挿入/削除) ペナルティを表す。

$$
\begin{aligned}
P_{i,j} = \quad & \min(P_{i-1,j-1} + s(a_i, t_j), && \#\text{置換} \\
& P_{i-1,j} + g, && \#\text{削除} \\
& P_{i,j-1} + g) && \#\text{挿入}
\end{aligned}
$$

例として、図 3 に、図 2 のペナルティ定義を用いてテキスト BABAC を検索キーワード ABC で検索する例を示す。格子上の数字は $P_{i,j}$ を表す。斜め線上の数字は置換ペナルティ、格子上以外の縦・横線上の数字は挿入/削除ペナルティを表している。太線はペナルティの合計が 2 以下の結果 (AB, ABA, ABAC, BAC, AC) を表している。

しかし、計算量は $O(mn)$ となり、検索対象が大きい場合、時間がかかりすぎる。そのため、計算量を低減するための手法が盛んに研究されている [1][2]。また、実用面を重視して、事前に検索対象を絞り込み DP マッチングの計算回数を低減するという手法も研究もされている。畑中ら [12] は文字

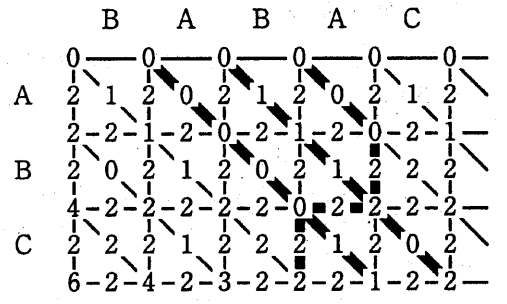


図 3: DP マッチング

列に対応付けられた整数値により類似度 (ハッシュ距離) 計算を行い検索対象を絞り込んでいる。ゲノムサイエンスの分野で、塩基配列やアミノ酸配列を類似検索するプログラムとしてがよく知られている FASTA では、固定長文字列での転置インデックスを用いて検索対象を絞り込んでいる [11]。

3.2 トライを用いる方法

Oflazer[3] はコード化した構文木を検索するためにトライ (TRIE) を用いた類似検索を行っている。これは Error-tolerant Recognition アルゴリズム [4] をトライに適用したものである。深さ優先でトライを探索し、ノードを辿る度に現在位置までの文字 (コード) 列と検索キーワードとの距離 (cut-off distance) を DP マッチングで計算し、それがある限度を越えたらその方向の探索を打ち切るという方法である。

図 4 に Error-tolerant Recognition アルゴリズムを示す[2]。t はペナルティの合計の限度を表す。q_i はトライ上のノードを表す。q_0 はトライのルートノードを表す。*cutdist* は cut-off distance で、以下のように定義される[3]。このアルゴリズムは第 2章で説明した辞書類似検索を扱うものである。

$$
\begin{aligned}
cutdist(m,k) &= \min_{l \leq i \leq u} P_{i,k} \\
l &= \max(1, k - \lfloor t/M \rfloor) \\
u &= \min(m, k + \lceil t/M \rceil)
\end{aligned}
$$

M はギャップのペナルティを表す。

[2]説明のために文献 [3] のものに少し変更を加えてある。
[3]cut-off distance の l と u の定義は文献 [3] と文献 [4] とで異なっている。ここでは後者の定義を採用した。

1. $push((空文字列, q_0))$

2. スタックが空になるまで以下の処理を繰り返す。

   (a) $pop((Y', qi))$

   (b) q_iから辿れる全てのノードと、そのときに通る枝のラベルVに対して以下の処理を行う。

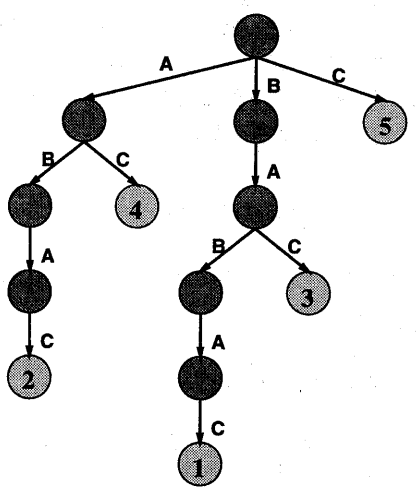
      i. $Y = concat(Y', V)$
      # 文字をつなげる

      ii. $k = length(Y)$ # 文字列の長さ

      iii. $cutdist(m, k) \leq t$ ならば $push(Y, q_j)$
      # 限度を越えていたら切り捨てる。

      iv. $P_{m,k} \leq t$ かつ q_j が終端ノードならば検索結果としてYを出力する。

図 4: Error-tolerant Recognition アルゴリズム



テキスト	B	A	B	A	C
ポインタ	1	2	3	4	5

図 6: Suffix Tree

ここで、Error-tolerant Recognition アルゴリズムを用いた簡単な例を図 5 に示す。左側は AAA, AB, BBA という 3 つの文字列から構成されるトライで、これから文字列 AAB をペナルティの合計が 1 以下という条件 ($t = 1$) で類似検索する過程を右側に示した。ギャップペナルティと置換ペナルティ(A → B,B → A) はどちらも 1 とする。灰色のボックスは cut-off distance を表す。BBA の方は、[B]→[B] とトライを辿るとこの時点で cut-off distance が 1 を越えてしまうので探索は打ち切られる。AAB,AB は探索は打ち切られることなく、葉までたどり着き、$P_{m,k}$(右下隅の数字) が $t(= 1)$ 以下なので検索結果として出力される。

4 本研究で提案する手法

本研究では前章で説明した Error-tolerant Recognition アルゴリズムを拡張してフルテキスト類似検索を実現した。

本章では、まず、辞書類似検索に特化している Error-tolerant Recognition アルゴリズムをフルテキスト類似検索に適用する方法について説明する。次に、トライ構造の欠点であるデータ領域の非効率性を解消するための方法として、Suffix Array[6] を用いた疑似トライ構造について説明する。最後に、定性的な特性を調べるため簡単な実験を行う。

4.1 Error-tolerant Recognition アルゴリズムのフルテキスト類似検索への適用

ここでは、Error-tolerant Recognition アルゴリズムのフルテキスト類似検索への適用について説明する。

フルテキスト検索のために、図 5 のような辞書的なトライではなく、図 6 に示すような Suffix Tree というフルテキスト検索ためのトライを用いる。suffix とはテキスト中のある位置からテキストの終了までを含む文字列であり、Suffix Tree とはテキストの全ての suffix を対象としたトライである。図 6 は、テキスト BABAC の全ての suffix (BABAC, ABAC, BAC, AC, C) に対してトライを作成したものである。終端ノード上の数字はポインタ (その suffix がテキストの何文字目から始まるか表す) である。

フルテキスト類似検索では検索キーワードと検索結果文字列の長さをそろえる必要はないので、Error-tolerant Recognition アルゴリズムで Suffix Tree を扱うことを考えた場合、アルゴリズムの検

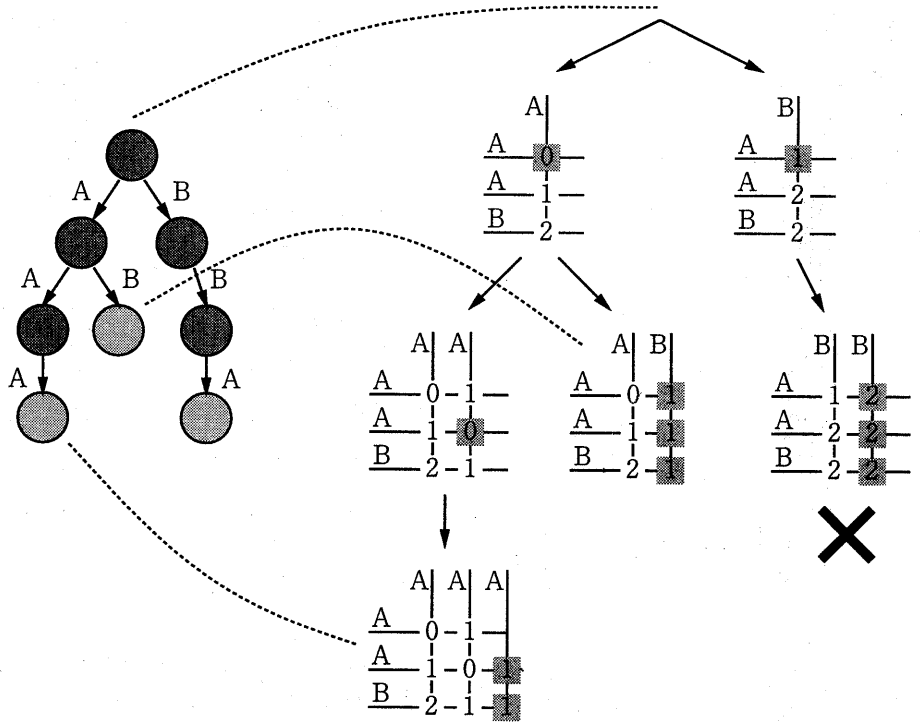


図 5: Error-tolerant Recognition の例

索結果出力条件「終端ノードか否かの判定」(図 4: 2. (b) iv.) は必要ない。例として、図 7 を用いて説明する。図 7 で Error-tolerant Recognition アルゴリズムを用い、Suffix Tree に対してペナルティの合計が 1 以下という条件で文字列 BA を類似検索してみる。ギャップペナルティと置換ペナルティ(A → B,B → A) はどちらも 1 とする。トライを [B] → [A] → [B] と辿った時点で BA(ペナルティ合計 0), B, BAA(ペナルティ合計 1) を検索結果として出力して欲しいのだが、Error-tolerant Recognition アルゴリズムでは終端ノードまで行かないと検索結果を出力してくれない (図 4: 2. (b) iv.)。結局さらに辿ることになり、ペナルティ合計が 1 を越え、途中で打ち切られてしまう。

そこで、単純にアルゴリズム (図 4) の 2. (b) iv. から「終端ノードか否かの判定」を取り除き以下のように書き換える。

> $P_{m,k} \leq t$ ならば検索結果として Y を出力する。

これにより終端ノードのあるなしに関わらずペナルティ合計がある限度以内の検索結果を出力でき、Suffix Tree に対しても使用できるアルゴリズムになる。

しかし、Suffix Tree の問題点としてデータ領域の非効率性が挙げられる。これは木構造のための付属情報が原因となっている。次節以降、この問題を Suffix Array を用いた疑似トライにより解決する。

4.2 Suffix Array の仕組み

Suffix Array[6] とは、ある検索キーワードを与えたときにそれがテキストのどの位置にあるのかを効率的に調べることができるフルテキスト検索のためのデータ構造である。

Suffix Array は、テキスト中の全て suffix (ある位置からテキストの終了までの文字列) をソートすることにより作成する。具体的な作成方法を以下に示す[4]。

[4]文献 [5] に C 言語による非常に短く分かりやすいプログラ

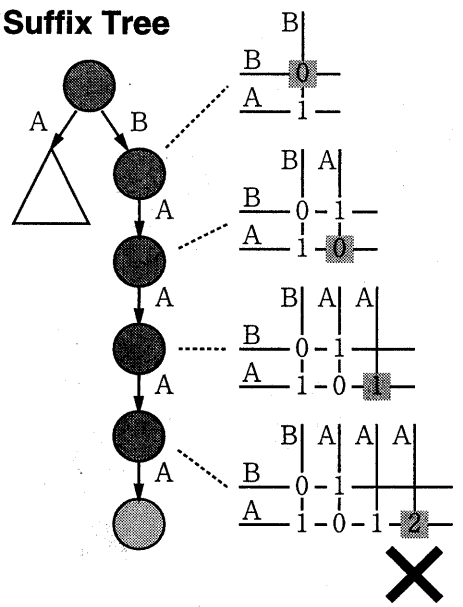


図 7: Error-tolerant Recognition アルゴリズムの不都合

No.	suffix
1	YAMASITATATUO
2	AMASITATATUO
3	MASITATATUO
4	ASITATATUO
5	SITATATUO
6	ITATATUO
7	TATATUO
8	ATATUO
9	TATUO
10	ATUO
11	TUO
12	UO
13	O

⇒

No.	suffix
2	AMASITATATUO
4	ASITATATUO
8	ATATUO
10	ATUO
6	ITATATUO
3	MASITATATUO
13	O
5	SITATATUO
7	TATATUO
9	TATUO
11	TUO
12	UO
1	YAMASITATATUO

図 8: Suffix Array

1. テキストに対するポインタの配列 (図 8 の左側の No. の列) を用意する。
2. そのポインタの配列を各ポインタが指す位置から始まる文字列でソートする。

図 8 中の文字列はポインタにより参照される suffix を分かりやすくするために表示したもので Suffix Array には含まれていない。図 8 の右側の No. の列だけが Suffix Array と呼ばれる。

Suffix Array は元テキストがあればポインタにより suffix を随時参照することができるので、実質的に辞書順に並べられた文字列として扱うことができる。それゆえ二分探索を用いて単純に検索 (前方一致検索) を行うことができる。

辞書順に並んでいるので、検索結果は "範囲" として表すことができる。例えば、図 8 の Suffix Array (右側の No. の列) で文字列 AT を検索した場合、配列の 3 つ目から 4 つ目の範囲 (つまり ATATUO と ATUO) が検索結果として返される。

テキストの大きさを n とした場合、必要なデータ領域は $O(n)$ となる。これは Suffix Tree と同じであるが、Suffix Array では必要なデータはポインタだけなので、実際のデータ領域には格段の開きがある。

我々は、以上に述べたような Suffix Array の作成と検索を行うシステム SUFARY を作成した。現在フリーソフトとして WWW 上で公開している [15]。

4.3 Suffix Array による疑似トライの実現

Suffix Array による疑似トライの実現方法について説明する。

まず、例として、テキスト BABAC に対する Suffix Array (図 9) を用いる。説明のために Suffix Array の各ポインタの指す suffix を縦書きに表示した。ここで、ABAC, AC の先頭の A、BABAC, BAC の先頭の B と 2 文字目の A をそれぞれまとめて一つのノードとみなせば図 6 と同等なトライ (Suffix Tree) となることに気付くであろう。

つまり、先頭から X 文字目までが共通な suffix を指すポインタの Suffix Array での範囲 (前節を参照) をトライ上の深さ X のノードとして扱うことで疑似的なトライ構造が実現できる。この疑似トライにおいて、検索文字列に従ってノードを辿るという操作は、検索文字列の先頭文字から始めて一文字ずつ増やしながら (つまり、徐々に検索範囲を狭めな

ムが挙げられている。

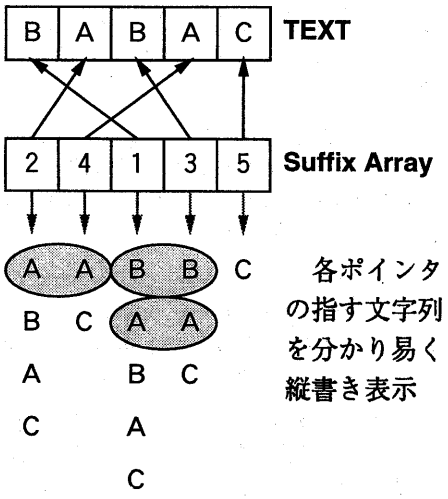


図 9: Suffix Array による疑似トライ

がら) Suffix Array を検索するという操作になる。検索の度に徐々に検索範囲が狭められていくので検索効率は悪くない。

例として、図 8 の Suffix Array を用いた疑似トライで、ATA を検索する。トライを A→T→A と辿るという操作は、Suffix Array で A, AT, ATA という文字列を順番に検索するという操作になる。その過程を図 10 に示す。まず A で検索して Suffix Array の一つ目 (AMASITA…) から四つ目 (ATUO) までの範囲を結果として得る。次にその範囲内で、AT を検索して三つ目 (ATATUO) から四つ目までの範囲を結果として得る。最後にその範囲内で、ATA を検索して最終的に三つ目から三つ目まで (つまり範囲内の要素は ATATUO のみ) を検索結果として得る。

4.4 実験

これまでに説明した Error-tolerant Recognition の Suffix Tree への適用と疑似トライによりフルテキスト類似検索システムを実現した。

システムの動作確認と定性的な特性を調べるために簡単な実験を行った。

大腸菌の塩基配列データ (約 4.7M バイト)[5] に対して、検索キーワードの文字数、ペナルティ合計の

[5] A, C, G, T の 4 種類の文字から構成されているテキストデータである。以下の URL から入手できる。
http://bsw3.aist-nara.ac.jp/GTC/mori/

		A	AT	ATA
2	AMASITATATUO	AMASITA…		
4	ASITATATUO	ASITATA…		
8	ATATUO	ATATUO	ATATUO	ATATUO
10	ATUO	ATUO	ATUO	
6	ITATATUO			
3	MASITATATUO			
13	O			
5	SITATATUO			
7	TATATUO			
9	TATUO			
11	TUO			
12	UO			
1	YAMASITATATUO			

図 10: 疑似トライでの検索

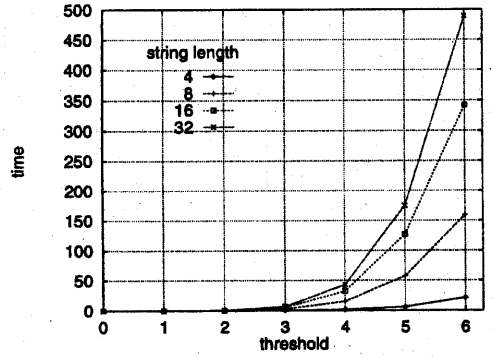
限度 (threshold) をいろいろ変えて検索時間を測定した。キーワードはランダムに作成されたものを用いた。これを 10 回行い、検索時間の平均をとった。図 11 に結果を示す。検索時間 (time) の単位は秒である。検索時間には検索結果の表示時間は含まれていない。使用したマシンは Sun Ultra 1(Solaris 2.5.1) である。実験 (1) は、ギャップペナルティが 1、異なる文字同士の置換ペナルティが 1 のときの、実験 (2) では、ギャップペナルティが 2、置換ペナルティが 1 のときの検索時間を示した。

ペナルティ合計の限度を増やすに連れて検索時間が指数的に増えていくことが分かる。また、ギャップペナルティの設定により検索時間にかなりの差が見られる。これは、ギャップペナルティを低く設定するとギャップ操作が頻繁に起こりトライの探索範囲が極端に広がるからである。今回のデータでは字種が 4 つ (A,C,G,T) であったが、字種が多ければ (当然、ペナルティの設定に依存するが) 置換操作による探索範囲の増大が見込まれる。

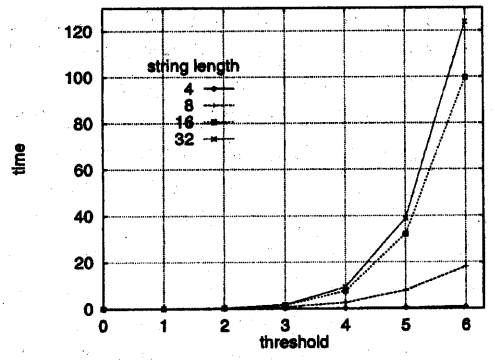
5 おわりに

データ領域の問題で実現が困難であったフルテキスト類似検索を Suffix Array を用いることにより実現できた。本研究で作成したシステムは、様々な分野で用いることのできるような汎用的なシステムになっている。SUFARY のホームページ [15] にて公開していく予定である。

今後、本システムを自然言語処理に応用していくつもりだが、言語情報のコード化とペナルティの設定を如何に行うかという点が課題となる。



実験 (1)　ギャップペナルティ = 1
　　置換ペナルティ　　= 1



実験 (2)　ギャップペナルティ = 2
　　置換ペナルティ　　= 1

図 11: 実験結果

参考文献

[1] Alfred V. Aho. "Algorithms for Finding Patterns in String", in: Jan V. Leeuwen, ed., "Handbook of Theoretical Computer Science", Elsevier Science Publishers, 1990.

[2] Gad M. Landau. "Fast Parallel and Serial Approximate String Matching", Journal of Algorithms, Vol.10, pp.157-169, 1989.

[3] Kemal Oflazer. "Error-tolerant Tree Matching", COLING-96: The 16th International Conference on Computational Linguistics, pp.860-864, 1996.

[4] Kemal Oflazer. "Error-tolerant Finite-state Recognition with Applications to Morphological Analysis and Spelling Correction", Computational Linguistics, Vol.22, No.1, pp.73-89, 1996.

[5] Kenneth W. Church. "You shall know a word by the company it keeps", NLPRS'95: Natural Language Proceeding Pacific Rim Symposium, pp.22-34, 1995.

[6] Udi Manber, Gene Myers. "Suffix Arrays: A New Method for On-line String Searches", 1st ACM-SIAM Symposium on Discrete Algorithms, pp.319-327, 1990.

[7] 青山典生, 兵藤安昭, 浅井泰博, 応江黔, 池田尚志. "形態素解析と意味コード化に基づく翻訳支援のための類似例文検索システム", 信学技報, NLC93-68, pp.39-45, 1994.

[8] 佐藤理史. "アナロジーによる機械翻訳", 共立出版株式会社, 1997.

[9] 隅田英一郎, 堤豊. "翻訳支援のための類似用例の実用的検索法", 電子情報通信学会論文誌 (D-II), Vol.J74-D-II, No.10, pp.1437-1447, 1991.

[10] 長尾真編. "自然言語処理", 岩波書店, 1996

[11] 中村春木, 中井謙太. "バイオテクノロジーのためのコンピュータ入門", コロナ社, 1995.

[12] 畑中念, 遠藤裕英. "日本語 OCR 文における英字・カタカナのスペル誤り訂正法", 情報処理学会論文誌, Vol.38, No.7, pp.1317-1327, 1997.

[13] 美馬秀樹, 隅田英一郎, 飯田仁. "類似検索を用いた情報検索システム", 言語処理学会第 2 回年次大会発表論文集, pp.113-116, 1996.

[14] 美宅成樹, 金久實 共編. "ヒトゲノム計画と知識情報処理", 培風館, 1995.

[15] 山下達雄, 熊谷俊高, 米沢恵司, 松本裕治. "Suffix Array を用いた高速文字列検索システム SUFARY", 1997.
http://cactus.aist-nara.ac.jp/lab/nlt/ss.html