人工知能学会研究会資料
SIG-FPAI-B402-08

文法圧縮を応用したハミング距離計算の高速化

Faster Calculating Algorithm for the Hamming Distance Using Grammar Compression

前田幸司[1*]　高畠嘉将[1]
田部井靖生[2]
坂本比呂志[1]
Koji Maeda[1]　Yoshimasa Takabatake[1]　Yasuo Tabei[2]　Hiroshi Sakamoto[1]

[1] 九州工業大学 情報工学府
[1] Graduate School of Computer Science and Systems Engineering, Kyushu Institute of Technology, Japan
[2] PRESTO さきがけ - 独立行政法人科学技術振興機構
[2] PRESTO, Japan Science and Technology Agency

Abstract: We propose a faster calculating algorithm for the Hamming distance using grammar compression. We search substrings which the Hamming distance with a query is less than d. ESP-index[10] is a self-index based on grammar compression. ESP-index builds a derivation tree by an input string. Matching node in a tree can ensure that some characters is same. Using the property, we reduce time that calculating of the Hamming distance.

1　はじめに

本論文では，文法圧縮を応用してハミング距離計算の高速化を行う．文法圧縮とは，与えられた文字列 S を一意に導出する CFG を構築する圧縮手法である．この手法は繰り返しの多いテキストに対しては効果的に圧縮を行うことができる．近年，文法圧縮を用いた索引 [1, 2, 3]，頻出パターン発見 [4] 等の様々な応用が提案されている．また [5] では文法圧縮を用いて、検索文字列 P に対して、ハミング距離 d 以下の文字列 S 中の部分文字列を列挙するアルゴリズムが提案されてきた．しかし [5] の方法では，圧縮データを復元してパターン P と一文字づつ比較を行うため，計算に時間を要していた．

ESP-index[10] は文字列を検索する際，文字列 S とパターン P で共通する木構造を発見することで高速に検索を行う．その性質を活かし，ハミング距離計算を行う際，文字列が一致している部分の復元を省くことにより高速化を行った．

2　準備

2.1　文法圧縮

文字の有限集合 Σ の要素からなる文字列全体の集合を Σ^* と定義する．生成規則 $X \to \alpha$ において X を変数と呼び，変数の集合を $V(G)$ と表す．生成規則の集合を辞書 $D(G)$ と呼び，$D(G)$ の生成規則の数を n とする．

文法圧縮とは，与えられた文字列 $S \in \Sigma^*$ を一意に導出する CFG G を作る圧縮手法である．このとき G は *admissible* であり，一意に文字列を導出する．したがって，任意の $\alpha \in (\Sigma \cup V(G))^*$ において，$X \to \alpha \in D(G)$ は高々一つしかない．また CFG はチョムスキー標準形の Straight line program(SLP) に制限し，生成規則は $X \to a (a \in \Sigma)$ もしくは $X_k \to X_i X_j (1 \leq i, j < k \leq |V(G)|)$ である．

例えば入力文字列 $S = aabbabab$ のときの生成規則と，CFG 木の一例を図 1 に示す．一般に生成規則数が少なくなるほど圧縮率は高くなるが，生成規則数を最小にする問題は NP 困難であることが証明されており [6]，Repair[7]，LCA[8] などの文法圧縮アルゴリズムで近似解を求めるアルゴリズムが複数提案されている．

*連絡先： 九州工業大学大学院 情報工学府
〒 820-8502 福岡県飯塚市川津 680 番 4
E-mail: k_maeda@donald.ai.kyutech.ac.jp

$A \to aa$
$B \to bb$
$C \to ab$
$D \to AB$
$E \to CC$
$F \to DE$

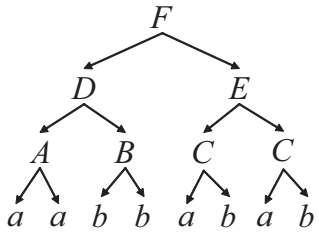

図 1: CFG の例

2.2 ESP-index

ESP-index[10] とは，文法圧縮を用いた索引構造である．

2.2.1 Edit Sensitive Parsing(ESP)

ここでは文法圧縮の構文木構築を ESP を用いて行う方法について述べる．ESP とは以下の通りである．

長さ u の文字列 S が与えられたとき，連続文字列 x_i と，同じ文字の連続を含まない文字列 w_i を用いて，S を $S' = w_1, x_1, w_2, x_2, \ldots, w_k, x_k, w_{k+1}$ に分解する．このとき，$w = \emptyset$ の可能性もある．この分割された S' において，x_i を Type1，w_i のうち $|w_i| \geq \lg^* |u|$ のものを Type2，Type1 でも Type2 でもないものを Type3 とする．ここで $\lg^* u = \{min\{i | \lg^{(i)} u \leq 1\}\}$ である．Type1 と Type3 については左優先で 2 文字づつ $S = s_1, s_2, \ldots, s_k (|s_i| = 2 (i < k))$ のように分割する．s_k は u が偶数のときは最後の 2 文字，奇数のときは最後の 3 文字となる．Type2 については，alphabet reduction[9] を用いて $S = s_1, s_2, \ldots, s_k (2 \leq |s_i| \leq 3)$ のように分割する．全ての Type において，$|s_i| = 2$ のとき $s_i = XY$ から $t_i \to XY$ が生成され，$|s_i| = 3$ のとき $s_i = XYZ$ から $t_i \to Xt'$，$t' \to YZ$ が生成される．

ESP で置き換えられた文字列に対して，繰り返し ESP を適用して $u = 1$ になるまで置き換えを行う．Type2 については以下の定理 2.1 が成り立つ．

定理 2.1 (Cormode and Muthukrishnan[9])
S が Type2 であり，alphabet reduction によって $S = s_1, s_2, \ldots, s_k$ と分割されたとする．もし $S\alpha$ が Type2 で $S\alpha = t_1, t_2, \ldots, t_n$ に分割されたとすると，$s_i = t_i (1 \leq i \leq k-5)$ が成り立つ．もし αS が Type2 で $\alpha S = t'_1, t'_2, \ldots, t'_m$ と分割されていると，$t'_{m-j} = s_{k-j} (0 \leq j \leq k - \lg^* u - 5)$ が成り立つ．

2.2.2 ESP-index を用いたパターン検索

ここでは，ESP-index におけるパターン検索の動作について述べる．長さ u の文字列 S と，長さ m の検索文字列 P に ESP を用いて生成された構文木を，それぞれ T_S，T_P と定義する．S 中に P が出現するとき，T_S 中に T_P を導出する部分木列が現れる．なぜなら，Type1 と Type2 では以下のように木が作られているからである．

Type1 では左優先で置き換えが行われることから，完全二分木の列が生成される．この完全二分木の葉はすべて連続文字列となっている．また完全二分木の列は左優先で置き換えられているので，完全二分木の個数は $O(\lg m)$ である，

Type2 で置き換えられた文字列はまた Type2 であり，ESP による置き換えを繰り返すので，定理 2.1 が再び成り立つ．従って定理 2.1 で分割が一致していない部分を抽出していくと，部分木の列が $O(\lg^* u \lg m)$ 個生成される．

従って，Type1 と Type2 では検索文字列 P からこれらの部分木の列を抽出して，T_S 中から探すことによって，ESP-index ではパターン検索を可能としている．Type3 の長さは $\lg^* u$ 以下なので，Type3 の葉の列を T_S 中から探す．

これらの検索すべき部分木の列を，evidence と呼ぶ．

また，evidence の中で符号化長が最も長いノードを core と呼ぶ．

定理 2.2 (Maruyama et al.[1])
P の evidence を T_S 中に埋め込めるか確認するのにかかる時間計算量は，$O(\lg^* u \lg m + \lg m)$ である．

定理 2.3 (Takabatake et al.[10])
ESP-index で検索文字列 P の出現位置を求めるのにかかる時間計算量は，$(\lg \lg n)(m + occ_c \lg m \lg u) \lg^* u$ である．ただし occ_c は T_s 中に存在する *core* の個数である．

定理 2.4 (Takabatake et al.[10])
ESP-index で検索文字列 P の出現位置を求めるのにかかる領域計算量は，$n \lg u + n \lg n + 2n + o(n \lg n)$ である．

定理 2.5 (Takabatake et al.[10])
ESP-index で部分文字列を復元するのにかかる時間計算量は，$(m + \lg u) \lg \lg n$ である．

定理 2.6 (Nakahara et al.[4])
長い周期に対して周期的でない S を仮定する．具体的には，$|\alpha| > \lg^* n$ となるような部分文字列 α^2 を含まない文字列である．S 中の任意の部分文字列 P に対して,P の最大 core の大きさが δm 以上になるような，定数 $\delta \geq \frac{1}{12(\lg^* n + 10)}$ が存在する．

2.3 ハミング距離

ハミング距離とは，長さの等しい文字列の中で，対応する位置にある異なった文字の個数である．例えば，$S_1 =$ wander と $S_2 =$ wonder は 2 番目に位置する文字が異なっており，ハミング距離は 1 となる．$S_3 =$ collect と $S_4 =$ correct は 3 番目と 4 番目に位置する文字が異なっており，ハミング距離は 2 となる．データ検索において，完全に一致する文字列だけでなく，このようなハミング距離のある文字列も求めることにより，似たゲノム構造の抽出や，タイポの検出などが可能である．

2.4 複数分類法と ESP-index を用いた曖昧検索

複数分類法とは，検索文字列 P を複数のブロックに分けてそれぞれを検索し，出現位置からハミング距離 d 以下になりうる位置を抽出するフィルタリング手法である．

複数分類法と ESP-index を組み合わせることにより，ハミング距離 d 以下の部分文字列検索を高速に行うことができる [5]．

STEP1 入力文字列 S の索引を ESP-index を用いて作成する．

STEP2 検索文字列 P を長さ $\lfloor \frac{|P|}{k} \rfloor$ の k(k は $k > d$ の整数) 個のブロック $P_1, P_2, \ldots, P_k$ に分割する．

STEP3 S を構築する際にできた辞書を用いて，全てのブロック P_i の構文木を構築し，S 中のすべてのブロック P_i の出現位置を求め，L_i に保存する．

STEP4 L_i に存在する位置 p を取り出し，L_j 中に位置 $p-(i-j)\times\frac{|P|}{k}$ $(1 \le j \le i-1)$, $p+j\times\frac{|P|}{k}$ $(i+1 \le j \le k)$ が存在するかを確かめる．このとき，存在しないブロックの個数が d 個以下だった場合，ハミング距離 d 以下になる可能性のある S の部分文字列として抽出する．この操作を，L_i 中に存在する位置 p の全てに対して行う．

STEP5 STEP4 の操作を L_1 から順に，L_{d+1} までに対して繰り返し行う．

STEP6 STEP5 までに抽出された候補位置の，一致しなかったブロック P_i と対応する S の部分文字列を比較し，詳細なハミング距離を求める．

STEP7 STEP6 を異なったすべてのブロックに対して行った後，求められたハミング距離の合計が d 以下だった場合，候補位置を答えとして出力する．

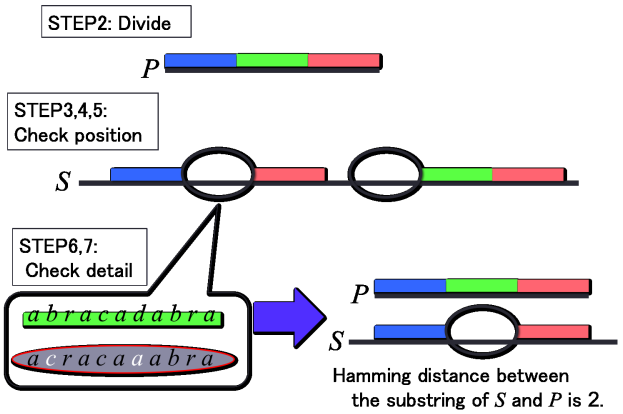

図 2: 複数分類法のイメージ

3 提案手法

2.4 節の手法における，STEP6 の高速化を図る．

3.1 ブロック内のハミング距離の求め方

文字列が異なったブロックに対応する T_S の部分に，P_i の evidence が埋め込めるかを確認する．evidence を埋め込められた場合，そのノードに含まれる終端文字列は一致することが明らかとなるので，それ以上探索を行わなくて良い．evidence を埋め込められなかった場合，そのノードに含まれる終端文字列が異なることから，終端文字列の復元を行った上，P_i と比較することによって正確なハミング距離を求める．

これまでブロック全体の文字列を復元するためにブロック長 $\lfloor \frac{|P|}{k} \rfloor$ だけのノードを探索する必要があった．しかしこの提案手法では，一致している evidence の探索を行わないことにより探索するノード数が減り，計算量が削減できる．

定理 3.1 Type1 の部分において，異なる終端文字数を d とすると，ハミング距離を求めるのに探索すべきノードの個数は $O(max(d, \lg m))$ である．

証明. Type1 は連続文字列であることを考慮すると，一致しているのか確認するノードは evidence の個数である $O(\lg m)$ であるが，一致していないノードに関してはさらに探索をして正確なハミング距離を確認する必要がある．したがって探索するノードの個数は，d が小さいとき，evidence の個数と同じ $O(\lg m)$．d が大きいとき，最大 $2d$ 個の葉を探索する必要があるため $O(d)$ となる．以上のことから，Type1 の部分の正確なハミング距離を求めるのに掛かる時間計算量は $O(max(d, \lg m))$ である． □

定理 3.2 Type2 の部分において，異なる終端文字数を d とすると，ハミング距離を求めるのに探索すべきノードの個数は $O(d \times \frac{m}{12(\lg^* n+10)})$ である．

証明. 複数分類法によって抽出された T_S の候補地に存在する T_P の evidence を求め，evidence と一致しなかった部分はすべて展開して詳細なハミング距離を確認する必要がある．一致しない evidence の個数は $O(d)$ であり，evidence が含む終端文字の個数は $O(\frac{m}{12(\lg^* n+10)})$ であるため，探索するノードの個数は $O(d \times \frac{m}{12(\lg^* n+10)})$ となる． □

定理 3.3 Type3 の部分において，異なる終端文字数を d とすると，ハミング距離を求めるのに探索すべきノードの個数は $O(\lg^* u)$ である．

証明. Type3 の部分はすべての終端文字を調べる必要がある．Type3 が含む終端文字列の個数は $\lg^* u$ 以下であるため，探索すべきノードの個数は高々$O(\lg^* u)$ である． □

4 実験

今回提案したアルゴリズムを実装し，[5] との計算時間を比較した．比較する対象は，①ハミング距離 d 以下の部分文字列を列挙する時間②今回改良を加えた 2.4 節の STEP6 に要した時間である．実験環境は，Intel Xeon CPU E7-8837 2.67GHz/1TB RAM，コンパイラは gcc4.4.7 を用いた．実験データは，入力文字列 S として Pizza&Chili Corpus(http://pizzachili.dcc.uchile.cl/) に存在する
Escherichia_Coli(107.5MB の，大腸菌の DNA データ)．検索文字列 P として Escherichia_Coli のランダムな位置から始まる長さ 100 文字から 1000 文字 (100 文字刻み) の部分文字列を，各長さにおいて 1000 個づつ用意した．索引サイズは 27MB，索引構築にかかった時間は 17.96sec である．また複数分類法におけるブロックの個数は，ESP-index が短い文字列の復元に時間が掛かるということを考慮し，求めるハミング距離 d を求めるのに最低限必要なブロック数である $d+1$ とした．ハミング距離 d は 2 から 4 までで実験を行った．各ハミング距離の設定で取得できた部分文字列の個数を表 1 示す．

①の 1 つの検索文字列あたりに要した時間を図 3,4,5 に示す．②の 1 つの検索文字列あたりに要した時間を図 6,7,8 に示す．

表 1: ハミング距離 dist 以下の部分文字列の出現回数

文字列長	d = 2	d = 3	d = 4
100	10158	11453	12278
200	6665	7732	8659
300	5217	6060	6969
400	4539	5177	5734
500	3853	4332	4809
600	3204	3560	3948
700	3256	3598	3911
800	2841	3094	3339
900	2739	2940	3143
1000	2520	2724	2900

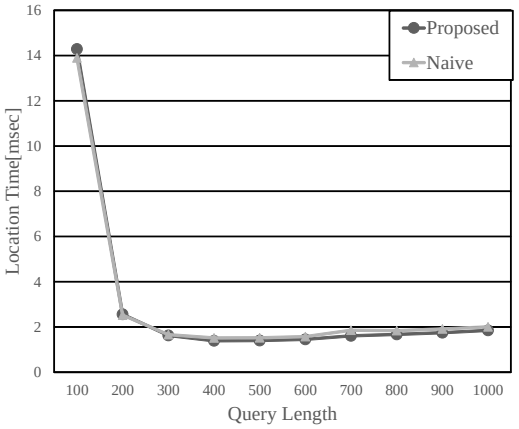

図 3: ハミング距離 2 の部分文字列の検索時間

図 3,4,5 より，今回改良を加えた部分に関して高速化できたこと確認できた．しかし図 6,7,8 より，ハミング距離 d 以下の部分文字列を列挙する動作のうち，提案手法による高速化できる部分はごく僅かであることが確認できた．提案手法は探索する evidence の数を大きく減らせるときにより計算量を減らすことができるため，検索文字列 P が長く，ハミング距離 d が小さいとき，より上手く働くと考えられる．

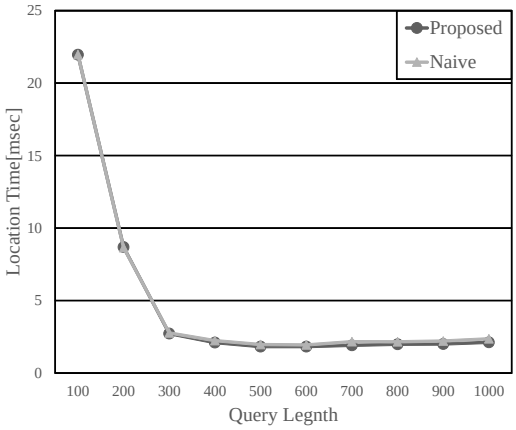

図 4: ハミング距離 3 の部分文字列の検索時間

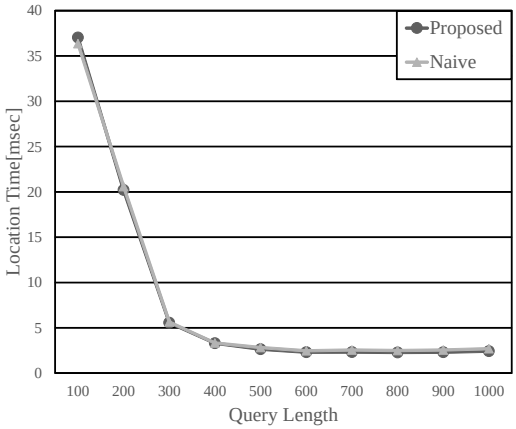

図 5: ハミング距離 4 の部分文字列の検索時間

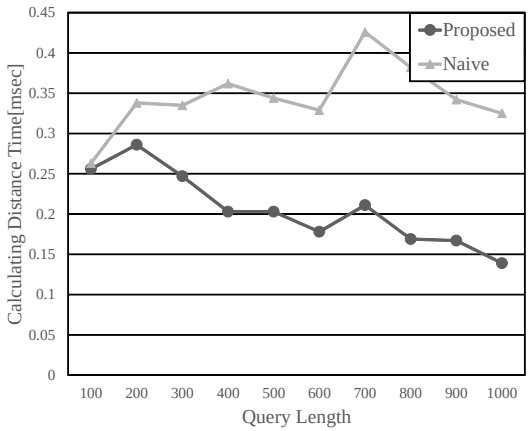

図 6: ハミング距離 2 の条件下で STEP6 に要した時間

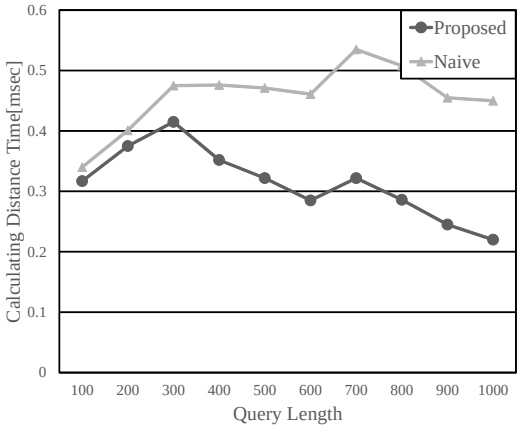

図 7: ハミング距離 3 の条件下で STEP6 に要した時間

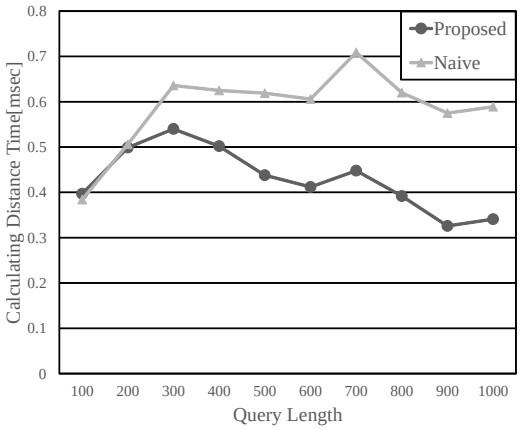

図 8: ハミング距離 4 の条件下で STEP6 に要した時間

5 おわりに

本稿では，ESP-index のパターン検索を応用し，ハミング距離計算の高速化を行った．この手法を用いることで巨大な検索文字列 P の場合でも STEP6 の部分では高速な動作が可能となった．一方でハミング距離 d 以下の部分文字列を列挙する問題に対しては，今回の提案手法による高速化の影響は少ないことも確認できた．

参考文献

[1] S. Maruyama, M. Nakahara, N. Kishiue, H. Sakamoto: ESP-index: A compressed index based on edit-sensitive parsing, *J. Discrete Algorithms*,Vol. 18,pp. 100–112 (2013)

[2] F.Claude,G.Navarro: Self-indexed grammar-based compression, *Fund. inform.*,Vol. 111(3),pp. 313–337(2011)

[3] T.Gagie,P.Gawrychowski,J.Karkkainen,Y.Nekrich,S.Puglisi: A faster grammar-based self-index, *LATA*,pp. 240–251(2012)

[4] M. Nakahara, S. Maruyama, T. Kuboyama, H. Sakamoto, Scalable Detection of Frequent Substrings by Grammar-Based Compression *IEICE Trans. on Information and Systems*,E96-D(3),pp. 457–464 (2013)

[5] 前田幸司，高畠嘉将，坂本比呂志，田部井靖生: 文法圧縮を応用したハミング距離の短い文字列列挙アルゴリズム第 92 回人工知能基本問題研究会, Nemuro (JPN), 2014

[6] E. Lehman, A. Shelat: Approximation algorithms for grammar-based compression. *In SODA* pp. 205–212,(2002)

[7] N.J. Larsson and A. Moffat: Offline Dictionary-Based Compression. *In DCC*, pp. 296–305(1999)

[8] S. Maruyama, H. Sakamoto, and M. Takeda: An Online Algorithm for Lightweight Grammar-Based Compression. *Algorithms* Vol. 5,pp.213–235(2012)

[9] G. Cormode, S. Muthukrishnan: The string edit distance matching problem with moves, *ACM Transactions on Algorithms*,Vol. 3 (1) (2007)

[10] Yoshimasa Takabatake, Yasuo Tabei, and Hiroshi Sakamoto: Improved ESP-index: a practical self-index for highly repetitive texts, *In SEA*,pp. 338-350 Copenhagen (DENMARK)(2014)