人工知能学会研究会資料
SIG-FPAI-B502-21

ストリームデータからの頻出パターンの近似発見

Approximate Frequent Pattern Discovery for Stream Texts

青山友紀 [1*]　高畠嘉将 [1] 坂本比呂志 [1]
Tomoki Aoyama[1]　Yoshimasa Takabatake[1]　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

Abstract: We propose an approximate frequent pattern discovery algorithm for stream texts. Frequent pattern discovery is to discover patterns that appear more than once in texts. Frequent pattern discovery is one of the basic techniques in the field of data mining. Grammar compression is a data compression for highly repetitive texts. Additionally, grammar compression extends an approximate fast frequent pattern discovery[1]. However, the algorithm is limited to offline settings. The algorithm requires input length for discovering approximate frequent patterns and does not support appending new texts. Therefore, we propose an online and space-efficient algorithm. It uses FOLCA[2] which is an online grammar compression. It requires compressed texts space and quickly supports appending new texts.

1　はじめに

本稿では，ストリームデータからの文法圧縮手法を用いた頻出パターンの近似発見を提案する．本稿で提案する手法はオフラインで動作する手法 [1] をもとに，ストリームデータから頻出パターンを発見できるように発展させ，さらにメモリ使用領域を削減したものである．本稿で述べる頻出パターンとは入力テキスト中に 2 回以上出現する部分文字列のことである．頻出パターン発見はデータマイニングの分野では基本的な手法のひとつであり，例えば，DNA の解析分野や，テキストの剽窃検出などに利用することができる．頻出パターンを素朴な手法により発見しようとすると，非常に時間がかかり，全文検索のような手法では時間計算量は $O(N^3)$ かかる．このときの N は入力テキストのサイズである．また，接尾辞配列を用いた手法も頻出パターン発見に利用できるが，この手法では領域計算量が $\Omega(N)$ であり使用領域が大きい．そこで，文法圧縮を用いた頻出パターン発見手法 [1] が提案された，この手法は頻出パターンの近似検出ではあるが，時間計算量は $O(N \lg^* N)$ であり，領域計算量は $O(N)$ である高速な手法である．しかし，この手法ではストリームデータに対応できておらず，また索引を構築し，すべてのデータを保持しているためメモリ使用領域はまだまだ大きい．そこで今回オンライン文法圧縮手法と定数領域での頻出アイテム発見手法を用いたストリームデータに対し省メモリで動作する頻出パターン発見のアルゴリズムを提案する．

*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: t_aoyama@donald.ai.kyutech.ac.jp

2　準備

本稿中では，文字の有限集合 Σ からなる文字列全体の集合を Σ^* と定義し，文字列 $S \in \Sigma^*$ の長さを $|S|$ と表現する．文字列 S 中の i 番目の文字と i から j 番目の文字列をそれぞれ $S[i]$，$S[i,j]$ と表記する．このとき，$1 \leq i \leq j \leq |S|$ である．また，対数の表記を $\lg = \log_2$ とし，$\lg^* N$ は $\lg^* N = \{min\{i | \lg^{(i)} N \leq 1\}\}$ を表すものとする．

2.1　頻出パターン発見問題

$|P| \leq |S|$ となる文字列 P をパターンと呼び，パターン P が文字列 S 中の部分文字列である場合，P が S 中に出現するという．また，$S[i,j] = P$ であるパターン P は文字列 S 中に複数回出現することがある．このときの i, j の組の個数を *occurrence* といい，i を P の *location* と呼ぶ．文字列 S に対するパターン P の *occurrence* が 2 以上であるときを P が S 中に頻出であると呼び，この頻出パターンを発見することが本稿で取り扱う頻出パターン発見問題である．今回は，文

法圧縮と ESP の特性を活かすことで頻出パターンを近似発見を可能にする．

2.2 文法圧縮

本稿上で述べる文法圧縮とは，与えられた文字列 S を一意に導出可能な CFG(context-free grammar) を構築する圧縮手法である．生成規則 $X \to \gamma$ について，X を非終端記号という．また，非終端記号の集合を V とし，生成規則の集合を D と呼び，D の生成規則の数を n とする．CFG 上で，Σ は終端記号の集合である．本稿では文字列を入力文字列としない限り $(\Sigma \cup V)^*$ と仮定する．CFG G は *admissible* であり，一意に文字列を導出することができるので，任意の $\alpha \in (\Sigma \cup V)^*$ に対して高々一つの $X \to \alpha \in D$ が存在している．このような生成規則は木構造で表すことができ，その木構造を構文木と呼ぶ．また，構文木において展開長とはある非終端記号から生成規則を用いて復号される文字列の長さのことである．CFG の木構造と生成規則の例を図 1 に示す．ここでの C の展開長は 4，A の展開長は 2 となる．

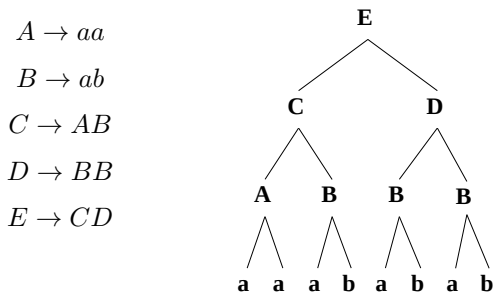

図 1: CFG の木構造と生成規則の例

文法圧縮とは文字列に対し，冗長な部分を生成規則としてまとめることで圧縮を行うので，生成規則が少ないほど圧縮率が高くなり，繰り返しの多いテキストに対し高い性能を誇る．しかし，この生成規則を最小とする問題は $NP-hard$[3] であることが知られており，その近似解を得る手法として Repair[4] や LCA[5] などの文法圧縮アルゴリズムが提案されている．今回は LCA をオンラインで動作可能にした FOLCA[2] と呼ばれる文法圧縮アルゴリズムを使用した．

2.3 Edit-Sensitive Parsing

Edit-Sensitive Parsing(ESP)[6] と呼ばれる分割のずれが保障された文字列分割法が存在する．これは，分割法でありながらそのアルゴリズムを利用することにより LCA や FOLCA などの文法圧縮アルゴリズムになることが知られている．このアルゴリズムで導出される構文木は同じ文字列からはほぼ同じ木構造を持つという性質がある．本稿では，このアルゴリズムを用いて構築される構文木のことを ESP 木と呼び，以下にその構築法を示す．

2.3.1 ESP 木構築アルゴリズム

ESP 木の構築には長さ N の入力文字列文字列を 2 文字か 3 文字のブロックに分割し、生成規則や非終端記号を生成し，それを 1 文字になるまで繰り返す．この分割は文字列を 3 つの type とみなすことで可能にしている．

type1 長さが 2 以上の同一した文字の連続文字列．

type2 同一した文字の連続文字列を含まない長さ $\lg^* N$ より長い文字列．

type3 同一した文字の連続文字列を含まない長さ $\lg^* N$ 以下の文字列．

以上の 3 つの type の内，type1，type3 については左から優先し，生成規則を作成していく．type2 に関してのみ alphabet reduction という方法を用いることにより，文字列を 2 文字か 3 文字に分割を行い，生成規則を作成する．alphabet reduction とは以下のような手順で行う分割のためのラベル付けである．

- *alphabet reduction*
  type2 の文字列 S が与えられた時，$S[i], S[i-1]$ を 2 進整数として考える．p を $S[i], S[i-1]$ の下位から見てはじめてビットが異なるポジション，$bit(p, S[i]) \in \{0, 1\}$ を $S[i]$ の p 番目のビットの値とする．この時，ラベル $L[i]$ を $L[i] = 2p + bit(p, S[i])$ と定義する．S は type2 の文字列なので連続同一文字を含まない．そのため，S から計算する新たなラベル列 $L = L[2]L[3]\cdots L[|S|]$ もまた同じラベルが連続して出現することはない．S に含まれる文字種類数を $\sigma = [S]$ とおく．この時，S に対して上の操作を行うと $[L] = 2\lg\sigma$ になる．この L に対して $[L] \leq \lg^* |S|$ になるまで操作を繰り返す． 最終ラベル L^* において下の式を満たす $S[i]$ を S の landmark と呼ぶ．

– $L^*[i] > max\{L^*[i-1], L^*[i+1]\}$ *(local maximum)*

– $L^*[i] < min\{L^*[i-1], L^*[i+1]\}$ *(local minimum)*

local maximum による landmark の決定後に local minimum による landmark を決定する．全ての landmark を決定後，landmark となった文字とその左の文字を優先的にペアにして非終端記号への置き換えを行う．

以上より，landmark の決定に関して以下の補題 1 が成り立つ．

補題 1 (Cormode and Muthukrishnan [6]) $S[i]$ の最近接の *landmark* は左に $\lg^* |S| + 5$ 文字，右に *5* 文字のみに依存し，決定される．

この補題 1 より，type2 の分割は，左に $\lg^* |S| + 5$ 文字，右に 5 文字を除いて一致することがいえる．つまり，ESP 木の特徴として，同じ文字列からは出現位置に関わらず左に $\lg^* |S| + 5$ 文字，右に 5 文字を除いて同一の構文木が構築される．

2.4 頻出パターン近似発見

文字列 S から構築された ESP 木を T とすると，T 中の内部ノードのラベルは非終端記号となるので，ある内部ノード V から導出される文字列 $S[n, m]$ が存在する．このときパターン $P = S[n, m]$ とする．T に出現する P から構築される部分木の中の最大共通部分木となるもの，つまり $S[n+k, m-l](k, l \geq 0)$ を常に導出する内部ノード v が存在するとき，この v のラベルを P に対する core と呼ぶ．この core のイメージ図を図 2 に示す．

また，この core の P に対する大きさについては以下の補題 2 が成り立つ．

補題 2 (M.Nakahara [1]) S の部分文字列 P の任意の出現に対し，P の *core* の長さは $\frac{|P|}{2\lg^2 |P|}$ 以上となる．

この補題 2 より，発見される頻出パターンの core の長さの下限が保障される．このことから頻出パターンの core を発見することは近似的に頻出パターンを発見することになる．したがって，Nakahara らの手法 [1] は ESP 木上に 2 回以上出現する非終端記号 (core) を発見することで，頻出パターンの近似発見を行っている．

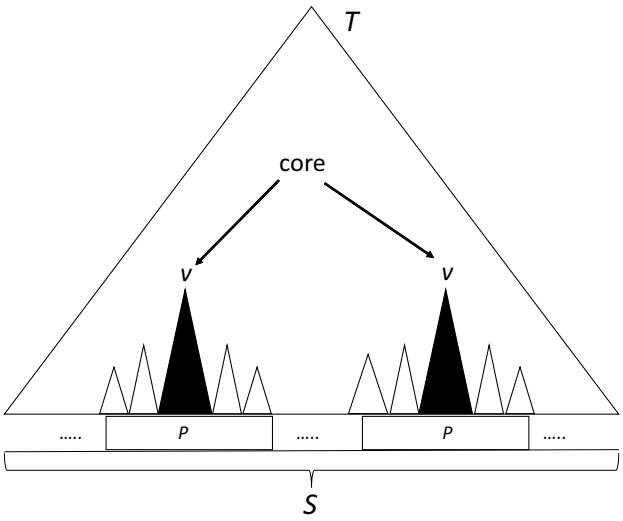

図 2: core のイメージ図

2.5 Simple Algorithm

Simple Algorithm[7] とは定数領域で頻度計算可能なアルゴリズムである．X_{item} は X に含まれるアイテムの数を表すものとすると，入力するアイテム列の長さを N，閾値を $\theta(0 < \theta < 1)$ とすると，$\frac{1}{\theta}$ 個のアイテムを保持することができ，これは最終的に出現頻度が $N\theta$ より大きいものが必ず含まれることが保障されている．頻度計算にかかる時間は $O(N)$ であり領域は $O(\frac{1}{\theta})$ である．

Simple Algorithm で保持しているアイテム集合を K としたときのアルゴリズムを以下に示す．

- 入力アイテムが K に含まれている場合
  そのアイテムの頻度を 1 増やす．
- 入力アイテムが K に含まれていない場合
  新たに入力されたアイテムの頻度を 1 とし，K に加える．このとき $K_{item} = \frac{1}{\theta}$ の場合に限り，K に含まれるすべてのアイテム頻度を 1 減少させる．減少したアイテムのうち頻度が 0 となったものを K より削除する．この削除操作には $O(\frac{1}{\theta})$ かかる．

3 提案手法

ESP 木をオンラインで構築可能なアルゴリズムである FOLCA[2] という文法圧縮手法を用いた．この手法により構築された ESP 木に出現するパターンの core の数をカウントすることにより頻出パターンの近似発見を行う．この core をカウントするために追加領域と

して生成規則と同数を保持することができる配列を用意した．これにかかる時間は $O(\frac{N \lg n}{\alpha \lg \lg n})$ かかり，領域は $(1+\alpha)n \lg(n+\sigma) + n(3 + \lg(\alpha n)) + n \lg N$ かかる．なお，ここでの N は入力テキスト長で，n は生成規則の数，σ は文字種類数であり，α はハッシュテーブルのための任意定数 $(0 < \alpha < 1)$ である．さらに本稿では頻出パターンの定義をテキスト中に 2 回以上出現するパターンとしているが，すべての頻出パターンを近似取得することは無駄が多い．そこで，今回はさらに省スペース化するために文法圧縮により生成された非終端記号を Simple Algorithm を用いることで定数領域で core の数をカウントする．Simple Algorithm を用いたアルゴリズムでは時間は $O(\frac{N \lg n}{\alpha \lg \lg n})$ で領域は $(1+\alpha)n \lg(n+\sigma) + n(3 + lg(\alpha n)) + O(\frac{1}{\theta})$ かかる．ここでの θ は Simple Algorithm で保持するアイテム数を決定するための任意の閾値 $(0 < \theta < 1)$ である．

4 実験

今回説明したアルゴリズムを実装を行い，実験を行った．実験に用いたアルゴリズムは，すべての core の出現頻度を取得する手法 (proposed)，Simple Algorithm で core の出現頻度を数える手法 (proposed-s)，オフラインで動作する手法 (offline)[1] を速度と領域について比較実験を行った．

4.1 実験環境と入力データ

今回実験を行った環境は，OS:CentOS 5.11，CPU:Intel Xeon E5504(2.00GHz, Quad Core)，Memory:144GB RAM，Compiler:gcc4.1.2 である．
また，今回実験に使用したデータは、英文テキストデータである einstein(400MByte)[1]と日本語版 wikipedia の編集履歴である jawiki(5120MByte)[2]であり，抽出する core の長さを 100 以上とし，proposed-s で保持するアイテムの数は einstein と jawiki でそれぞれ 1,000 個と 1,000,000 個とした．

4.2 実験結果

4.2.1 速度比較

速度比較の表を表 1 に示す．

einstein と jawiki ともに offline のアルゴリズムが提案手法より速くなっている．これは，オンラインで動作する FOLCA に比べオフラインで動作するアルゴリズムである従来手法が速いためであり，また，jawiki での proposed-s は proposed に比べ遅くなっているが，これは Simple Algorithm でのアイテム消去に時間がかかっているためである．

表 1: 速度比較の表

手法	入力データ	時間 (s)
proposed-s	einstein	503
	jawiki	29,183
proposed	einstein	503
	jawiki	7,943
offline	einstein	121
	jawiki	1437

4.2.2 領域比較

領域比較の表を表 2 に示す．この領域は，memusage コマンドにより消費されたヒープ領域の上限を取得した．

表 2: 領域比較の表

手法	入力データ	領域 (MByte)
proposed-s	einstein	16
	jawiki	2,276
proposed	einstein	32
	jawiki	4,382
offline	einstein	2,000
	jawiki	5,120

einstein と jawiki ともに提案手法のアルゴリズムが offline のアルゴリズムより領域を削減している．さらに proposed-s では proposed の 50% 程度の領域での頻出パターン発見が可能となっている．

5 おわりに

本稿では，ESP 木の性質を用いて，ストリームデータから頻出なパターンを近似発見するアルゴリズムを提案した．時間計算量は $O(\frac{N \lg n}{\alpha \lg \lg n})$ で，領域は $(1+\alpha)n \lg(n+\sigma) + n(3 + \lg(\alpha n)) + O(\frac{1}{\theta})$ である．提案手法では従来の手法より省メモリな手法となっているが，速度が遅くなっている．速度に関しては，アイテム保持数を調整することや，Simple Algorithm のアイテム消去を $O(1)$ で処理可能なアルゴリズム [8] が提案されているので，これを用いることである程度の速度の改善が望めると考えられる．また，領域に関しては定数領域で動作する文法圧縮アルゴリズムが提案されている [9]．このアルゴリズムは使用領域を定数領域に抑え

[1] http://dumps.wikimedia.org/jawiki/20151202
[2] http://pizzachili.dcc.uchile.cl/repcorpus/real

て文法圧縮を行う．そこで，この手法を用いることで，定数領域での頻出パターンの近似発見が可能になると考えられる．

参考文献

[1] 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):457-464, (2013)

[2] S. Maruyama, Y. Tabei, H. Sakamoto, K. Sadakane: Fully-Online Grammar Compression. *20th String Processing and Information Retrieval Symposium (SPIRE)*, 218-229, (2013)

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

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

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

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

[7] R. Karp, S. Shenker, C. Papadimitriou: A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst. 28*, E96-D(3):51-55, (2003)

[8] S. Iwasaki, H. Sakamoto: ストリーム中の頻出アイテム発見のための Simple Algorithm の高速化. 人工知能基本問題研究会 *98*, 32-35, (2015)

[9] S. Maruyama, Y. Tabei: Fully Online Grammar Compression in Constant Space. *DCC*, 173-182, (2014)