

省領域な lexicographic parse 構築アルゴリズム

クップル ドミニク

概要

可逆圧縮の分野において、LZ77 分解 (Lempel–Ziv 77 分解) は有名な手法の一つである。1977 年に発明されて以来、圧縮率といった複数のパラメタを工夫した派生手法が考案されている。近年提案された lex-parse も LZ77 の派生手法の一つであるが、実用的な計算手法としての提案には至らなかった。この論文では、PLCP 配列と ϕ 配列を用いて lex-parse を効率的に計算する方法を解析する。

キーワード： 可逆圧縮, Lempel–Ziv 77 分解, 少領域のアルゴリズム

1 序論

可逆圧縮は、圧縮のうち、復元前の文字列を完全な形で取り出すことが可能な圧縮方法である。文字列の可逆圧縮における一つの専門的分野、LZ77 分解 (Lempel–Ziv 77 分解) がよく知られている。LZ77 は任意文字列 T に対し、 T を項に分解する。このとき、LZ77 分解は任意の項がその項の開始位置より前に出現を持つように分解する。各項を長さとの出現の開始位置の二つ組に表現できる。ただし、後半を参照先と呼ぶ。すなわち、項を二つ組で表現した場合でも、元文字列を復元できる。二つ組の列による圧縮表現は、その表現が元文字列より小さい場合に与えられる。同様の圧縮表現は LZ77 以外にも多数研究されており、一般的に *macro scheme* [10] と呼ばれている。しかし、多くの macro scheme においては、ある項の参照先がその項の開始位置よりも前であることは保証されない。このように、参照先の選択肢が多いことが原因で、最小な macro scheme の計算問題は NP 完全に属している [10]。その問題への取り組みとして、LZ77 を始めとした様々な近似方法が提案された。lcpcomp [2], plcpcomp [3], や LZRR [8] はその最たる例である。今回取り上げた lex-parse [7, 節 VI] もその手法の一つである。lex-parse ではある項を定めるとき、この項の開始位置から始まる接尾辞 $T[i..]$ と、その接尾辞 $T[i..]$ から見て辞書式順序で直前にある接尾辞 $T[j..]$ を選び、 j を参照先、 $T[i..]$ と $T[j..]$ の共通接頭辞を項の長さに設定する。lex-parse は $O(n)$ 時間と $O(n \lg n)$ bit 領域で構築可能であるが、本論文ではこれをより少ない領域で計算することを目的とする。より詳細には、PLCP 配列 [9, 11] と ϕ 配列 [6] によって、lex-parse は $|\phi| + 3n + o(n)$ bits で計算できる。ただし n は入力文字列の長さ、 $|\phi|$ は ϕ の bit 領域を示す。実際には ϕ 配列をすべて利用することはなく、計算時は $r \lg n$ bit の sparse ϕ 表現で充分である。ただし r は Burrows–Wheeler transform [1] の文字の連の個数を示す。現在、最も領域面で優れた ϕ 配列の構築アルゴリズムは $O(n \lg \sigma)$ bit を必要とする [4]。

2 アルゴリズム

提案されたアルゴリズム [7] は接尾辞配列 SA, 逆接尾辞配列 ISA と最長共通接頭辞配列 LCP を利用し、 $O(n)$ 時間と $O(n)$ words 領域で lex-parse を計算できる。SA は T の接尾辞を辞書式順序で整列した配列である。すなわち、 $SA[i]$ は辞書式順序で i 番目の接尾辞の開始位置を示す。接頭辞の開始位置の定義領域は $[1..n]$ であるため、SA は順列、ISA は SA である。各 $i \in [2..n]$, $LCP[i]$ は $T[SA[i]..]$ と $T[SA[i-1]..]$ の最長共通接頭辞の長さを格納する。

このデータ構造を前処理で計算したのち、以下のアルゴリズムで lex-parse を計算できる。まず、 $T[1.. \max(1, LCP[ISA[i]])]$ を最初の項とし、各の項 F_x の開始位置 $i_x = 1 + \sum_{y=1}^{x-1} |F_y|$ に対して、 F_x の長さを $\max(1, LCP[ISA[i_x]])$ と

する．以後，次の項 F_{x+1} に対して再帰的に繰り返す．上記の手順により，項の長さを決定できる．各の項 F_x の参照先は $SA[ISA[i_x] - 1]$ で定める，ただし i_x は F_x の開始位置である．各の配列を定数時間でアクセスできる条件のもとで，全部の項を線形時間で計算できる．そのために上記の3つの整数配列が必要である．

3 2つの整数配列

領域を効率的に削減すべく，整数配列を見直す．SA, ISA, LCP という3つの配列の代わりに， ϕ と PLCP 配列のみを用いる． ϕ は i 番目の要素に $SA[ISA[i] - 1]$ を格納し，PLCP は i 番目の要素に $LCP[ISA[i]]$ を格納する． $\max(1, PLCP[i_x])$ は i_x の開始位置を持つ項 F_x の長さ， $\phi(i_x)$ は F_x の参照先を定める．

3.1 既存の実装

この2つの数列を元に *lex-parse* を計算するアルゴリズムは *tudocomp* [2] (<https://tudocomp.github.io/>) で実装された．図1は *lex-parse* の計算過程と計算量の例を示す．*tudocomp* で，*divsufsort* と呼ばれる接尾辞配列構築アルゴリズムでテキストから SA を計算し，SA から ϕ を計算できる．そのあと， ϕ をもとに PLCP を計算する [11]．計算した各配列は単純な 32-bit 整数配列に格納されている．次に，2つの配列をそれぞれ bit-optimal 整数配列に移動されて，最後に，分割アルゴリズム (Factorization) が実行された．入力は約 400 MB の *Pizza&Chili corpus* (<http://pizzachili.dcc.uchile.cl/>) のデータセット *para* である．図によると，分割アルゴリズムは前処理より早く，少領域である．しかしながら，前処理は圧倒的に遅く，膨大な領域を取るといった弱点がある．よって，この論文の目的は前処理を最適化し，前処理の要求データを削減することである．

3.2 改善

改善の方針は2つある．1つ目の改善点は，考案したアルゴリズムは線形的にテキストを読み込むから，整数配列で表現する PLCP は不要であり，上記で説明した $2n$ -bit 表現で充分である．2つ目は， ϕ を圧縮表現することである．大まかに， $BWT[ISA[i]] = BWT[ISA[i] - 1]$ ならば， $\phi[i] = \phi[i - 1] + 1$ となる [5, Lemma 3.3]．ただし， $BWT[i] = T[SA[i] - 1 \bmod |T|]$ は Burrows–Wheeler transform である． r が BWT の文字の連の個数とすると， ϕ は長さ r の整数配列と長さ n の bit vector で表現できる．

参考文献

- [1] M. Burrows and D. J. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.
- [2] P. Dinklage, J. Fischer, D. Köppl, M. Löbel, and K. Sadakane. Compression with the *tudocomp* framework. In *Proc. SEA*, volume 75 of *LIPICs*, pages 13:1–13:22, 2017. doi: 10.4230/LIPICs.SEA.2017.13.
- [3] P. Dinklage, J. Ellert, J. Fischer, D. Köppl, and M. Penschuck. Bidirectional text compression in external memory. In *Proc. ESA*, pages 41:1–41:16, 2019. doi: 10.4230/LIPICs.ESA.2019.41.
- [4] K. Goto and H. Bannai. Space efficient linear time Lempel–Ziv factorization for small alphabets. In *Proc. DCC*, pages 163–172, 2014. doi: 10.1109/DCC.2014.62.

- [5] J. Kärkkäinen and D. Kempa. LCP array construction in external memory. *ACM Journal of Experimental Algorithmics*, 21(1):1.7:1–1.7:22, 2016. doi: 10.1145/2851491.
- [6] J. Kärkkäinen, G. Manzini, and S. J. Puglisi. Permuted longest-common-prefix array. In *Proc. CPM*, volume 5577 of *LNCS*, pages 181–192, 2009. doi: 10.1007/978-3-642-02441-2_17.
- [7] G. Navarro, C. Ochoa, and N. Prezza. On the approximation ratio of ordered parsings. *IEEE Trans. Inf. Theory*, 67(2):1008–1026, 2021. doi: 10.1109/TIT.2020.3042746.
- [8] T. Nishimoto and Y. Tabei. LZRR: LZ77 parsing with right reference. In *Proc. DCC*, pages 211–220, 2019. doi: 10.1109/DCC.2019.00029.
- [9] K. Sadakane. Succinct representations of lcp information and improvements in the compressed suffix arrays. In *Proc. SODA*, pages 225–232, 2002.
- [10] J. A. Storer and T. G. Szymanski. Data compression via textural substitution. *J. ACM*, 29(4):928–951, 1982. doi: 10.1145/322344.322346.
- [11] N. Välimäki, V. Mäkinen, W. Gerlach, and K. Dixit. Engineering a compressed suffix tree implementation. *ACM Journal of Experimental Algorithmics*, 14, 2009. doi: 10.1145/1498698.1594228.

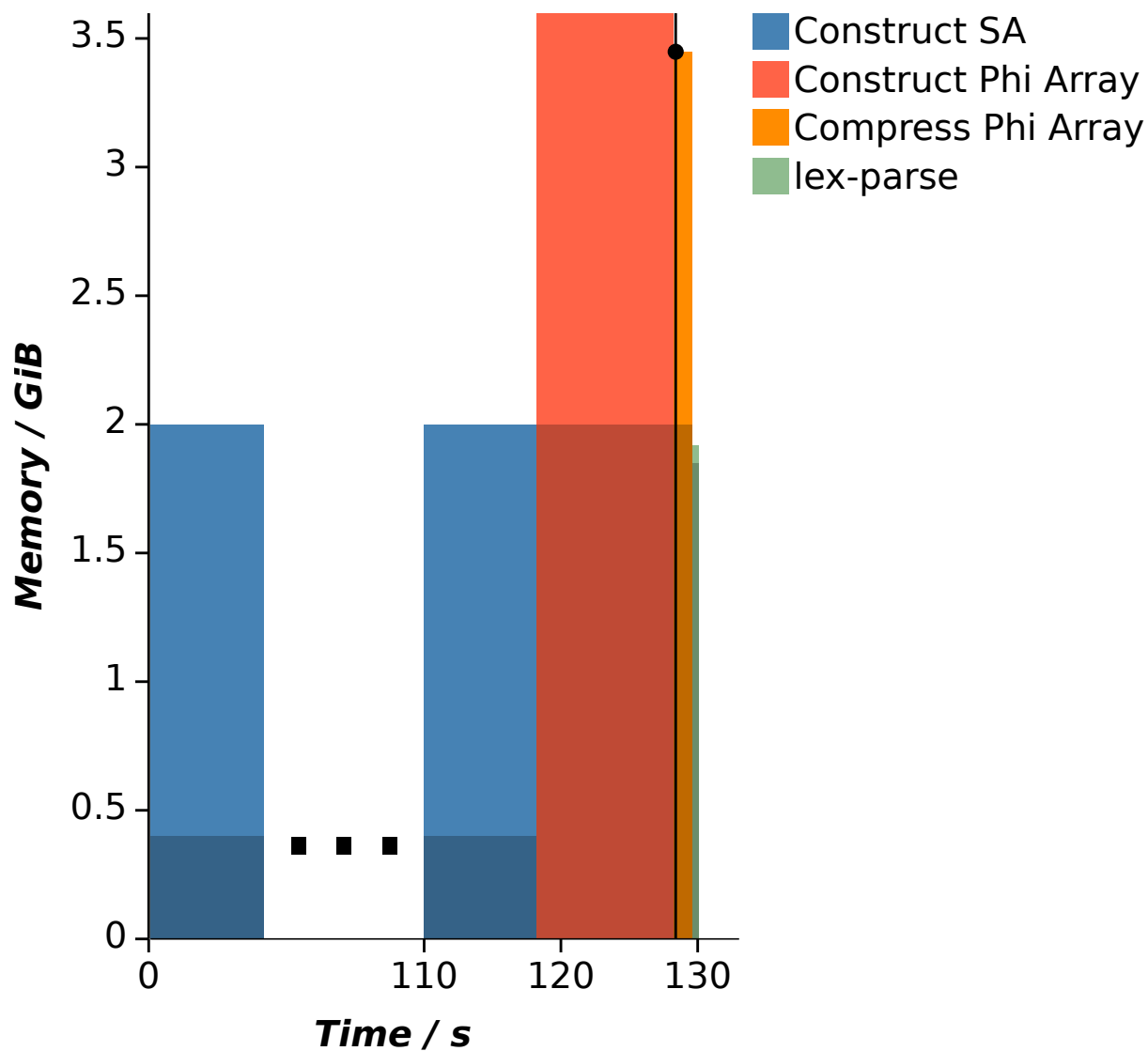


図 1: tudocomp で実装された lex-parse の計算過程と計算量の図である．入力は Pizza&Chili の PARA データセットを用いた．構築は過程ごとに分け，長方形の面積は各過程における時間と領域を示す．各の長方形は濃淡で 2 つの部分長方形で上下に分かれている．下部の濃い部分は直前の過程から引き継ぐメモリ領域である．薄い部分は新しく割り当てた領域である．