Journal of Discrete Algorithms 32 (2015) 64–68

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Approximate pattern matching in LZ77-compressed texts [☆]

Travis Gagie [a,*,1], Paweł Gawrychowski [b,2], Simon J. Puglisi [a,3]

[a] Department of Computer Science, University of Helsinki, Finland
[b] Max-Planck-Institut für Informatik, Saarbrücken, Germany

ARTICLE INFO

Article history:
Available online 14 October 2014

Keywords:
Compressed approximate pattern matching
LZ77

ABSTRACT

Suppose we want to support approximate pattern matching in a text $T[1..n]$ whose LZ77 parse consists of z phrases. In this paper we show how, given that parse, we can preprocess T in $\mathcal{O}(z \log n)$ time and space such that later, given a pattern $P[1..m]$ and an edit distance k, we can perform approximate pattern matching in $\mathcal{O}(z \min(mk, m + k^4) + occ)$ time and $\mathcal{O}(z \log n + m + occ)$ space, where occ is the size of the output.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Amir and Benson [1] first posed the problem of compressed approximate pattern matching in 1992: given a compressed representation of a text $T[1..n]$, an uncompressed pattern $P[1..m]$ and an integer $k \geq 0$, return the occ distinct positions in T that are start-points (or, symmetrically, end-points) of substrings within edit distance k of P — but without decompressing T. In this paper we are interested in compressed approximate pattern matching in LZ- and grammar-compressed texts.

Kärkkäinen, Navarro and Ukkonen [13] showed how to perform approximate pattern matching in LZ78- or LZW-compressed texts in $\mathcal{O}(z'mk + occ)$ time, where z' is the number of phrases in the LZ78 or LZW parse of T. Bille, Fagerberg and Gørtz [2] showed how any algorithm for uncompressed approximate pattern matching can be adapted for compressed approximate pattern matching in LZ78- or LZW-compressed texts, and thus improved the bound to $\mathcal{O}(z' \min(mk, m + k^4) + occ)$. Very recently, Gawrychowski and Straszak [12] gave a bound of $\mathcal{O}(z'm^{1/2}k^3 + occ)$.

Bille et al. [3] extended Bille, Fagerberg and Gørtz's technique to straight-line programs (SLPs). They showed how, given an SLP for T with r rules, we can perform approximate pattern matching in $\mathcal{O}(r \min(mk, m + k^4) + r \log n + occ)$ time. (The $r \log n$ term can be removed, and their approach simplified somewhat, by using a data structure due to Gąsieniec et al. [10].) Tiskin [21] gave a bound of $\mathcal{O}(rm \log m + occ)$ and noted that Kärkkäinen, Navarro and Ukkonen's algorithm can be made to run in $\mathcal{O}(rmk^2 + occ)$ time.

We further extend Bille, Fagerberg and Gørtz's technique to LZ77. We show how, if the LZ77 parse of T consists of z phrases then, given that parse, we can preprocess T in $\mathcal{O}(z \log n)$ time and space such that later, given P and k, we can perform approximate pattern matching in $\mathcal{O}(z \min(mk, m + k^4) + occ)$ time and $\mathcal{O}(z \log n + m + occ)$ space. We believe this

[☆] A preliminary version of this paper [8] was presented at the 22nd International Symposium on Algorithms and Computation (ISAAC '11). Part of this research was done while the first author was at Aalto University, the second author was at the University of Wrocław and the third author was at King's College London.

[*] Corresponding author.
E-mail address: travis.gagie@gmail.com (T. Gagie).

[1] Supported by the Academy of Finland grant 268324.
[2] Supported by NCN grant 2011/01/D/ST6/07164, 2011–2014.
[3] Supported by the Academy of Finland grant 258308.

http://dx.doi.org/10.1016/j.jda.2014.10.003
1570-8667/© 2014 Elsevier B.V. All rights reserved.

to be interesting because, first, the preprocessing time is dominated whenever m and k are reasonably large; second, the size of the LZ77 parse is a lower bound on the size of the smallest SLP for T, which is in turn a lower bound on the sizes of the LZ78 and LZW parses. Our model throughout is a word RAM with $\Omega(\log n)$-bit words.

2. Preliminaries

2.1. LZ77, LZ78 and LZW

Ziv and Lempel's LZ77 [25] and LZ78 [26] and Welch's LZW [24] are all popular text compression algorithms that work by partitioning their input into phrases. In the LZ77 parse, each phrase $T[i..j]$ is chosen such that $T[i..j-1]$ occurs in $T[1..j-2]$ but $T[i..j]$ does not occur in $T[1..j-1]$ (unless $j=n$). The leftmost occurrence of $T[i..j-1]$ in T is called the source of the phrase $T[i..j]$. In the LZ78 parse, each phrase $T[i..j]$ is chosen such that $T[i..j-1]$ occurs as a phrase in the parse of $T[1..i-1]$ but $T[i..j]$ does not (again, unless $j=n$). The LZW parse is a more complicated variant of the LZ78 parse, with similar properties.

Notice that the number of phrases in the LZ77 parse is never more than the number in the LZ78 parse and can be much less; the same is also true with respect to the LZW parse. For example, if T is unary then its LZ77 parse consists of two phrases, $T[1]$ and $T[2..n]$, whereas the LZ78 and LZW parses always consist of $\Omega(n^{1/2})$ phrases. On the other hand, while it is not difficult to build an $\mathcal{O}(z')$-space data structure that supports random access to T in $\mathcal{O}(\log\log n)$ time, Verbin and Yu [22] showed that any $(z\log^{\mathcal{O}(1)} n)$-space data structure must use $\mathcal{O}(\log^{1-\epsilon} n)$ time in the worst case, for any constant $\epsilon > 0$.

2.2. Straight-line programs

An SLP is a context-free grammar in Chomsky normal form that generates only one string. They are widely studied because many popular compression algorithms, including LZ78, LZW and Larsson and Moffat's RePair [16], can be viewed as producing SLPs. For an introduction, we refer readers to Lohrey's survey [17]. An SLP is called balanced if, in the corresponding parse tree, whenever a node has two subtrees they differ in size by at most a constant factor.

Rytter [20] and Charikar et al. [4] independently proved that any SLP for T has at least z rules, and showed how to turn T's LZ77 parse into a balanced SLP for T with $\mathcal{O}(z\log(n/z))$ rules in $\mathcal{O}(z\log n)$ time. (Their algorithms assumed that no phrase overlaps its sources, but Gawrychowski [11] removed this assumption.) Since the parse tree for a balanced SLP for T has height $\mathcal{O}(\log n)$, if we store the size of each non-terminal's expansion, then we can support extraction of any substring of length ℓ in $\mathcal{O}(\log n+\ell)$ time, nearly matching Verbin and Yu's lower bound.

Theorem 1. *(See [20,4].) Given the LZ77 parse of $T[1..n]$, in $\mathcal{O}(z\log n)$ time we can build a balanced SLP for T with $\mathcal{O}(z\log(n/z))$ rules, where z is the number of phrases in the parse.*

Bille et al. [3] showed how, given any (possibly unbalanced) SLP for T with r rules, in $\mathcal{O}(r)$ time we can build a data structure that supports extraction of any substring of length ℓ in $\mathcal{O}(\log n+\ell)$ time. Gąsieniec et al. (see also [5]) showed how we can build in $\mathcal{O}(r)$ time a data structure that supports extraction of any length-ℓ prefix or suffix of a non-terminal's expansion in $\mathcal{O}(\ell)$ time.

Theorem 2. *(See [9,10].) Given an SLP for T with r rules, in $\mathcal{O}(r)$ time we can build a data structure that supports extraction of any length-ℓ prefix or suffix of a non-terminal's expansion in $\mathcal{O}(\ell)$ time.*

2.3. Bookmarking balanced SLPs

In a previous paper [7] we showed how, given a balanced SLP with r rules for T, the size of each non-terminal's expansion, and a position i in T, we can store $\mathcal{O}(1)$ extra words such that later, given a length ℓ, we can extract $T[i-\ell..i+\ell]$ in $\mathcal{O}(\log\log n+\ell)$ time. We then say that i is bookmarked.

To bookmark i, we find the lowest common ancestor of the $(i-\log n)$th and $(i+\log n)$th leaves of the corresponding parse tree; find the lowest node v_1 on the rightmost path in u's left subtree, whose subtree includes the $(i-\log n)$th leaf; find the lowest node v_2 on the leftmost path in u's right subtree, whose subtree includes the $(i+\log n)$th leaf; and store the non-terminals X_1 and X_2 at v_1 and v_2 and the offset of the ith leaf in whichever of their subtrees contains it. This take a total of $\mathcal{O}(\log n)$ time.

Since the SLP is balanced, v_1 and v_2 have height $\mathcal{O}(\log\log n)$. It follows that, given $\ell\le\log n$, we can extract $T[i-\ell..i+\ell]$ from the expansions of X_1 and X_2 in $\mathcal{O}(\log\log n+\ell)$ time; given $\ell>\log n$, we can extract $T[\max(i-\ell,1)..\max(i+\ell,n)]$ as usual in $\mathcal{O}(\log n+\ell)=\mathcal{O}(\ell)$ time.

If we add a new rule $X\to X_1X_2$, we obtain a balanced SLP with at most $r+1$ rules for $T[i-\log n..i+\log n]$. We can compute the length of X's expansion in $\mathcal{O}(1)$ time by summing the lengths of X_1's and X_2's expansion. Therefore, in $\mathcal{O}(\log\log n)$ time we can store another $\mathcal{O}(1)$ extra words such that later, given ℓ, we can extract $T[i-\ell..i+\ell]$ in $\mathcal{O}(\log\log\log n+\ell)$ time. Recursing $\log^* n$ times takes a total of $\mathcal{O}(\log n)$ time and $\mathcal{O}(\log^* n)$ extra words, and allows us to later extract $T[i-\ell..i+\ell]$ in $\mathcal{O}(\ell)$ time regardless of ℓ.

Lemma 1. *(See [7].) Given a balanced SLP for $T[1..n]$ with r rules, the size of each non-terminal's expansion, and a position i in T, in $\mathcal{O}(\log n)$ time we can store $\mathcal{O}(\log^* n)$ extra words such that later, given ℓ, we can extract $T[i-\ell..i+\ell]$ in $\mathcal{O}(\ell)$ time.*

As a historical aside, we note that in the conference version [8] of this paper, we showed how to bookmark a new compressed representation of T, which we called T's block graph; only later did we realize how to bookmark balanced SLPs. Very recently we learned that, in the unpublished full version [9, Theorem 3] of their paper, Gąsieniec et al. showed how to bookmark arbitrary positions with their data structure; by chance, they also used the term bookmark. Their technique for bookmarking a position i takes extra space proportional to the depth of the ith leaf of the corresponding parse tree, however, which is $\Omega(\log n)$ for most values of i. Nevertheless, it seems possible to prove the main result of this paper using their result instead of our own Lemma 1.

2.4. Uncompressed approximate pattern matching

There is a large body of literature on approximate pattern matching in uncompressed texts, a survey of which is beyond the scope of this paper. For an introduction, we refer readers to Navarro's survey [19]. As mentioned in Section 1, Bille, Fagerberg and Gørtz's technique works with any algorithm for uncompressed approximate pattern matching. For concreteness, they considered an algorithm by Landau and Vishkin [15] that runs in $\mathcal{O}(nk)$ time, and an algorithm by Cole and Hariharan [6] that runs in $\mathcal{O}(nk^4/m+n)$ time. When dovetailed, these yield an algorithm that takes $\mathcal{O}(\min(nk,(n/m)(m+k^4)))$ time and $\mathcal{O}(m)$ space.

Theorem 3. *(See [15,6].) Given a text $T[1..n]$, a pattern $P[1..m]$ and an integer $k \geq 0$, we can find the occ distinct positions that are start-points of substrings of T within edit distance k of P, in $\mathcal{O}(\min(nk,(n/m)(m+k^4)))$ time and $\mathcal{O}(m)$ space.*

An improvement to Theorem 3 would imply corresponding improvements to Bille, Fagerberg and Gørtz's result, Bille et al.'s result and our result in this paper. For example, if we can perform uncompressed pattern matching in $\mathcal{O}(f(n,m,k))$ time, then we can perform approximate pattern matching in LZ78 or LZW texts in $\mathcal{O}(z' f(2(m+k),m,k)+occ)$ time.

2.5. BFG and BLRSSW algorithms

A key idea behind Bille, Fagerberg and Gørtz's technique is that no substring longer than $m+k$ characters can be within edit distance k of P. Therefore, if we extract and scan the $m+k$ characters to either side of each phrase boundary in an LZ parse, then we find the start-points of all substrings of T within edit distance k of P that could end at or cross phrase boundaries. Such approximate occurrences of P are called primary occurrences. Notice we include all approximate occurrences of P that start at most $m+k$ characters before the next phrase boundary, even when they end before that boundary, because algorithms for uncompressed approximate pattern matching may not tell us those occurrences' end-points.

Bille, Fagerberg and Gørtz showed how, given the LZ78 or LZW parse of T, in $\mathcal{O}(z')$ time we can build a data structure that supports extraction of any length-ℓ prefix or suffix of a phrase in $\mathcal{O}(\ell)$ time. Another way to do this is to convert the parse into an SLP for T, which takes $\mathcal{O}(z')$ time, then apply Theorem 2 to that SLP. With either of these data structures, we can extract the $m+k$ characters to either side of each phrase boundary in a total of $\mathcal{O}(z'm)$ time (since we can assume $k<m$). Applying Theorem 3 to each of these substrings takes a total of $\mathcal{O}(z' \min(mk, m+k^4))$ time. We can assume the start-points in each substring are returned in sorted order, since we can sort them in $\mathcal{O}(m)$ time and space.

Lemma 2. *(See [2].) Given the LZ78 or LZW parse of a text T, a pattern $P[1..m]$ and an edit distance k, we can find all the start-points of primary occurrences of P in T in sorted order in $\mathcal{O}(z' \min(mk, m+k^4))$ time, where z' is the number of phrases in the parse.*

An approximate occurrence of P that starts more than $m+k$ characters before the next phrase boundary is called secondary. Assume we have already found and sorted all the primary occurrences' start-points in each phrase of T, and all the secondary occurrences' start-points in each phrase of $T[1..i-1]$. Now we want to find all the secondary occurrences' start-points in the next phrase $T[i..j]$. This is easy when $i=j$, so assume $i<j$. By the definition of the parse, $T[i..j-1]$ occurs as a phrase in the parse of $T[1..i-1]$; let $T[h..h+j-i-1]$ be this earlier phrase.

For $0 \leq q \leq j-i-m-k$, there is a secondary occurrence starting at $T[i+q]$ if and only if there is an occurrence (primary or secondary) starting at $T[h+q]$, because $T[i+q..i+q+m+k-1] = T[h+q..h+q+m+k-1]$. We need not concern ourselves with $T[j-m-k+1..j]$ because any occurrence starting there is, by definition, primary and not secondary. It follows that we can build a sorted list of all the secondary occurrences' start-points in $T[i..j]$ using $\mathcal{O}(1)$ time per start-point.

Therefore, by induction, once we have all the start-points of primary occurrences of P in T, we can find all the start-points of P's secondary occurrences in $\mathcal{O}(z'+occ)$ time. Combining this with Lemma 2 yields the following theorem:

Theorem 4. *(See [2].) Given the LZ78 or LZW parse of a text T, a pattern $P[1..m]$ and an integer $k \geq 0$, we can find the occ start-points of substrings of T within edit distance k of P in $\mathcal{O}(z' \min(mk, m+k^4)+occ)$ time, where z' is the number of phrases in the parse.*

As mentioned in Section 1, Bille et al. considered the case in which we are given an SLP for T with r rules and asked to perform approximate pattern matching in it. Following Miyazaki, Shinohara and Takeda [18], Bille et al. observed that if $X \to YZ$ then all the approximate occurrences of P in X's expansion are either in Y's expansion, in Z's expansion, or in the substring consisting of the $m+k-1$ characters to either side of the boundary between them. It follows that, if we extract and scan the $m+k$ character to either side of the boundary between the expansions of the symbols on the right-hand side of each rule, then we can find all the approximate occurrences of P in T using an additional $\mathcal{O}(r + occ)$ time.

Bille et al. used their random-access data structure, described in Section 2.2, to extract the desired substrings in a total of $\mathcal{O}(r(\log n + m))$ time. If we apply Theorem 2 instead, however, we use only $\mathcal{O}(rm)$ time. Applying Theorem 3 to each of these substrings takes a total of $\mathcal{O}(r \min(mk, m + k^4))$ time.

Theorem 5. *(See [3].) Given an SLP for T with r rules, a pattern P and an integer $k \geq 0$, we can find the occ start-points of substrings of T within edit distance k of P in $\mathcal{O}(r \min(mk, m + k^4) + occ)$ time.*

3. Our algorithm

We follow Bille, Fagerberg and Gørtz's example and consider separately the tasks of finding the primary and secondary occurrences of P in T. Given the LZ77 parse of T, we first apply Theorem 1 to it and store the size of the expansion of each non-terminal in the resulting balanced SLP for T, which takes a total of $\mathcal{O}(z \log n)$ time. We then apply Lemma 1 to the SLP once for each phrase in the LZ77 parse of T, with the parameter i set to the position of the first character in the phrase; this also takes a total of $\mathcal{O}(z \log n)$ time.

We thus obtain in $\mathcal{O}(z \log n)$ time a data structure that supports extraction of the ℓ characters to either side of any phrase boundary in $\mathcal{O}(\ell)$ time. Later, given P and k, we extract the $m+k$ characters to either side of each phrase boundary and apply Theorem 3 to the resulting substrings, which takes a total of $\mathcal{O}(z \min(mk, m + k^4))$ time and returns all the start-points of primary occurrences of P in T in sorted order.

Our data structure takes $\mathcal{O}(z \log n)$ space, applying Theorem 3 to each substring in turn takes $\mathcal{O}(m)$ space and the number of start-points of primary occurrences is at most occ, so we use $\mathcal{O}(z \log n + m + occ)$ space for this step. Again, we can assume the start-points of primary occurrences in each substring are returned in sorted order.

Lemma 3. *Given the LZ77 parse of a text $T[1..n]$, we can preprocess T in $\mathcal{O}(z \log n)$ time and space such that later, given a pattern $P[1..m]$ and an edit distance k, we can find all the start-points of primary occurrences of P in T in sorted order in $\mathcal{O}(z \min(mk, m + k^4))$ time and $\mathcal{O}(z \log n + m + occ)$ space, where z is the number of phrases in the parse and occ is the number of occurrences of P in T.*

When preprocessing the LZ77 parse of T, we create a sorted list L_{sources} of the start-points of the phrases' sources, with a pointer from each phrase in the parse to its source's start-point in L_{sources}. (Recall from Section 2.1 that the source of phrase $T[i..j]$ is the leftmost occurrence of $T[i..j-1]$ in T.) This takes $\mathcal{O}(z \log z) \subseteq \mathcal{O}(z \log n)$ time and space.

When we are given P and k, we first apply Lemma 3 to obtain a sorted list $L_{\text{primaries}}$ of the primary occurrences' start points. We then add a pointer from each source's start-point in L_{sources} to that start-point's successor in $L_{\text{primaries}}$, in a total of $\mathcal{O}(z + occ)$ time and space. Finally, we use $L_{\text{primaries}}$ and L_{sources} to build a sorted list $L_{\text{secondaries}}$ of P's secondary occurrences, as we describe next.

Assume that, at some point, we have already:

- built the prefix of $L_{\text{secondaries}}$ consisting of all the secondary occurrences' start-points before a phrase $T[i..j]$;
- added a pointer from each source's start-point in L_{sources} to that start-point's successor in $L_{\text{secondaries}}$, if its successor is less than i;
- stored a pointer s to i's successor in L_{sources}.

We then find all the secondary occurrences' start-points in $T[i..j]$. This is easy when $i = j$, so assume $i < j$.

We follow the pointer to the start-point h of $T[i..j]$'s source in L_{sources}; follow the pointers to h's successors in $L_{\text{primaries}}$ and $L_{\text{secondaries}}$; and enumerate all the start-points in $L_{\text{primaries}}$ and $L_{\text{secondaries}}$ that are in $T[h..h+j-i-m-k]$. Notice that, unlike in Section 2.5, h may not be the start-point of a phrase itself, which complicates our bookkeeping.

As before, however, for $0 \leq q \leq j-i-m-k$ there is a secondary occurrence starting at $T[i+q]$ if and only if there is an occurrence (primary or secondary) starting at $T[h+q]$. Again, we need not concern ourselves with $T[j-m-k+1..j]$ because any occurrence starting there is primary and not secondary.

For each start-point $h+q$ we enumerate in $T[h..h+j-i-k]$, we append a start-point $i+q$ to $L_{\text{secondaries}}$; advance s past all the start-points in L_{sources} less than or equal to $i+q$; and add pointers from those start-points in L_{sources} to $i+q$ in $L_{\text{secondaries}}$. This takes $\mathcal{O}(1)$ time for each start-point in $T[h..h+j-i-k]$ or that s passes in L_{sources}. Notice we must append $i+q$ to $L_{\text{secondaries}}$ immediately, before continuing the enumeration, in case phrase $T[i..j]$ overlaps its source.

When we have finished the enumeration, we have

- appended to $L_{\text{secondaries}}$ all the secondary occurrences' start-points in $T[i..j]$;
- added a pointer from each source's start-point in L_{sources} to that start point's successor in secondaries, if its successor is between i and j;
- advanced s to $(j+1)$'s successor in L_{sources}.

Therefore, by induction, once we have all the start-points of primary occurrences of P in T, we can find all the start-points of P's secondary occurrences in $\mathcal{O}(z + occ)$ time and space. Combining this with Lemma 3 yields the main result of this paper:

Theorem 6. *Given the LZ77 parse of a text $T[1..n]$, we can preprocess T in $\mathcal{O}(z \log n)$ time and space, where z is the number of phrases in the parse, such that later, given a pattern $P[1..m]$ and an integer $k \geq 0$, we can find the occ start-points of substrings of T within edit distance k of P in $\mathcal{O}(z \min(mk, m + k^4) + occ)$ time and $\mathcal{O}(z \log n + m + occ)$ space.*

4. Postscript

Wandelt and Leser [23] later but independently developed Bille, Fagerberg and Gørtz's [2] technique and applied it to a variant of LZ77 that achieves good compression on repetitive datasets in practice and supports fast random access (see [14]). Their experiments indicate that compressed approximate pattern matching in, e.g., databases of human genomes, can significantly outperform the naïve approach of decompression followed by uncompressed approximate pattern matching.

References

[1] A. Amir, G. Benson, Efficient two-dimensional compressed matching, in: Proceedings of the Data Compression Conference (DCC), 1992, pp. 279–288.

[2] P. Bille, R. Fagerberg, I.L. Gørtz, Improved approximate string matching and regular expression matching on Ziv–Lempel compressed texts, ACM Trans. Algorithms 6 (1) (2009).

[3] P. Bille, G.M. Landau, R. Raman, K. Sadakane, S.R. Satti, O. Weimann, Random access to grammar-compressed strings, in: Proceedings of the 22nd Symposium on Discrete Algorithms (SODA), 2011, pp. 373–389.

[4] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, A. Shelat, The smallest grammar problem, IEEE Trans. Inf. Theory 51 (7) (2005) 2554–2576.

[5] F. Claude, G. Navarro, Improved grammar-based compressed indexes, in: Proceedings of the 19th Symposium on String Processing and Information Retrieval (SPIRE), 2012, pp. 180–192.

[6] R. Cole, R. Hariharan, Approximate string matching: a simpler faster algorithm, SIAM J. Comput. 31 (6) (2002) 1761–1782.

[7] T. Gagie, P. Gawrychowski, J. Kärkkäinen, Y. Nekrich, S.J. Puglisi, A faster grammar-compressed self-index, in: Proceedings of the 6th Conference on Language and Automata Theory and Applications (LATA), 2012, pp. 240–251.

[8] T. Gagie, P. Gawrychowski, S.J. Puglisi, Faster approximate pattern matching in compressed repetitive texts, in: Proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC), 2011, pp. 653–662.

[9] L. Gąsieniec, R.M. Kolpakov, I. Potapov, P. Sant, Real-time traversal in grammar-based compressed files, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.6639.

[10] L. Gąsieniec, R.M. Kolpakov, I. Potapov, P. Sant, Real-time traversal in grammar-based compressed files, in: Proceedings of the Data Compression Conference (DCC), 2005, p. 458.

[11] P. Gawrychowski, Pattern matching in Lempel–Ziv compressed strings: fast, simple and deterministic, in: Proceedings of the 19th European Symposium on Algorithms (ESA), 2011, pp. 421–432.

[12] P. Gawrychowski, D. Straszak, Beating $\mathcal{O}(nm)$ in approximate LZW-compressed pattern matching, in: Proceedings of the 24th International Symposium on Algorithms and Computation (ISAAC), 2013, pp. 78–88.

[13] J. Kärkkäinen, G. Navarro, E. Ukkonen, Approximate string matching on Ziv–Lempel compressed text, J. Discrete Algorithms 1 (3–4) (2003) 313–338.

[14] S. Kuruppu, S.J. Puglisi, J. Zobel, Relative Lempel–Ziv compression of genomes for large-scale storage and retrieval, in: Proceedings of the 17th Symposium on String Processing and Information Retrieval (SPIRE), 2010, pp. 201–206.

[15] G.M. Landau, U. Vishkin, Fast parallel and serial approximate string matching, J. Algorithms 10 (2) (1989) 157–169.

[16] N.J. Larsson, A. Moffat, Offline dictionary-based compression, in: Proceedings of the Data Compression Conference (DCC), 1999, pp. 296–305.

[17] M. Lohrey, Algorithmics on SLP-compressed strings: a survey, Groups Complex. Cryptol. 4 (2) (2012) 241–299.

[18] M. Miyazaki, A. Shinohara, M. Takeda, An improved pattern matching algorithm for strings in terms of straight-line programs, in: Proceedings of the 8th Symposium on Combinatorial Pattern Matching (CPM), 1997, pp. 1–11.

[19] G. Navarro, A guided tour to approximate string matching, ACM Comput. Surv. 33 (1) (2001) 31–88.

[20] W. Rytter, Application of Lempel–Ziv factorization to the approximation of grammar-based compression, Theor. Comput. Sci. 302 (1–3) (2003) 211–222.

[21] A. Tiskin, Semi-local string comparison: algorithmic techniques and applications, Technical report, arXiv:0707.3619v19, 2013.

[22] E. Verbin, W. Yu, Data structure lower bounds on random access to grammar-compressed strings, in: Proceedings of the 24th Symposium on Combinatorial Pattern Matching (CPM), 2013, pp. 247–258.

[23] S. Wandelt, U. Leser, String searching in referentially compressed genomes, in: Proceedings of the Conference on Knowledge Discovery and Information Retrieval (KDIR), 2012, pp. 95–102.

[24] T. Welch, A technique for high-performance data compression, Computer 17 (6) (1984) 8–19.

[25] J. Ziv, A. Lempel, A universal algorithm for sequential data compression, IEEE Trans. Inf. Theory 23 (3) (1977) 337–343.

[26] J. Ziv, A. Lempel, Compression of individual sequences via variable-rate coding, IEEE Trans. Inf. Theory 24 (5) (1978) 530–536.