Engineering Augmented Suffix Sorting Algorithms

Felipe A. Louza[1], Guilherme P. Telles[1] (advisor), Simon Gog[2] (co-advisor)

[1]Programa de Pós-Graduação em Ciência da Computação
Instituto de Computação
Universidade Estadual de Campinas (UNICAMP) – Campinas, SP – Brazil

[2]Institute of Theoretical Informatics
Karlsruhe Institute of Technology (KIT) – Germany

{louza,gpt}@ic.unicamp.br, gog@kit.edu

Abstract. *Strings are prevalent in Computer Science and algorithms for their efficient processing are fundamental in various applications. The results introduced in this work contribute with theoretical improvements and practical advances in building full-text indexes. Our first contribution is an in-place algorithm that computes the Burrows-Wheeler transform and the longest common prefix (LCP) array. Our second contribution is the construction of the suffix array augmented with the LCP array in optimal time and space for strings from constant size alphabets. Our third contribution is a set of algorithms to construct full-text indexes for string collections in optimal theoretical bounds. This work is an extended abstract of the Ph.D. thesis of the first author.*

1. Introduction

A large portion of the persistent information produced by people is formed by texts. Texts also form a large part of the digital world around us, which includes the Internet and its billions of webpages, genomic data from lots of different organisms and individuals, and very large non-relational datasets. Algorithms for processing texts (strings) are under intensive research, mostly to enable dealing efficiently with huge amounts of data that change on a daily basis.

A full-text index is a data structure built over a string that allows solving many string processing problems, from exact and approximated string matching to more involving tasks (some books, *e.g.* [Gusfield 1997, Ohlebusch 2013, Navarro 2016], present examples of many different applications in distinct areas). Full-text indexes are particularly useful in searching for substrings in strings that don't have a well defined notion of words, like DNA sequences, protein sequences, and East Asian languages, applications where other data structures as inverted lists fail.

The suffix array [Manber and Myers 1993], the longest common prefix (LCP) array and the Burrows-Wheeler transform (BWT) [Burrows and Wheeler 1994] are central in recent developments in full-text indexing.

The suffix array is an array of integers that holds the position of every suffix of a string in lexicographic order. Many problems in strings may be solved by means of one or more binary searches on suffix arrays. The LCP array holds the length of the longest common prefix between two consecutive sorted suffixes. The information provided by

i	SA	LCP	BWT	sorted suffixes
0	6	0	a	$
1	5	0	n	a$
2	3	1	n	ana$
3	1	3	b	anana$
4	0	0	$	banana$
5	4	0	a	na$
6	2	2	a	nana$

Figure 1. Suffix array, LCP array and BWT for T=banana$.

the LCP array can be used to accelerate searches on suffix arrays and allows other optimizations. The BWT is a reversible transformation of a string into another one that is easier to compress. The BWT of a string T has a deep relation with the suffix array of T, which allows constructing indexes for a string that are both compact and efficient. Figure 1 shows the suffix array (SA), the LCP array, the BWT and the sorted suffixes for the string $T = banana\$$.

Suffix array construction, also known as suffix sorting, is a well studied problem that may be solved in linear time (see [Puglisi et al. 2007, Kärkkäinen 2016] for reviews). A remarkable suffix array construction algorithm was presented by Nong [Nong 2013], which runs in $O(n)$ time and uses $O(\sigma \log n)$ bits of workspace, for a string of length n and alphabet size σ. The workspace is the additional space beyond input and output. When the alphabet size is constant ($\sigma = O(1)$), which is the case in many practical applications, for instance with molecular biology data and texts in Western languages, Nong's algorithm is optimal since it runs in linear time using constant workspace.

The LCP array can be computed in linear time given the string T and its suffix array as input using $O(n \log n)$ bits of workspace (*e.g.* [Kasai et al. 2001, Manzini 2004, Kärkkäinen et al. 2009]). Alternatively, the LCP array can be computed during the suffix array construction, also in linear time using $O(n \log n)$ bits of workspace [Fischer 2011]. The LCP array can be represented using less than $n \log n$ bits. Some alternatives for compressing the LCP array store its values in text order, building an array known as permuted LCP (PLCP) [Sadakane 2002, Kärkkäinen et al. 2009, Fischer 2010]. However, most applications will require the LCP array itself, and will convert the PLCP to the LCP array [Gog and Ohlebusch 2013].

The BWT can be computed easily in linear time given the string T and its suffix array as input. However, such approach needs $n \log n$ additional bits to compute and store the suffix array. Alternatively, the BWT can be computed directly from the text in linear time using $O(n \log \sigma \log \log_\sigma n)$ bits of workspace [Okanohara and Sadakane 2009]. The most space-efficient BWT construction algorithm is due to Crochemore *et al.* [Crochemore et al. 2015]: it builds the BWT in place — *i.e.*, replacing the input string with the BWT — in quadratic time and constant workspace for unbounded alphabets.

Another important problem related to suffix sorting is the construction of text indexes for string collections [Ohlebusch 2013]. This task may be performed in linear time using a common suffix sorting algorithm over the concatenation of all strings.

Given d strings $T_1, T_2, \ldots, T_d$ of total length N, there are two common alternatives to concatenate them into a single string T^{cat}. The first alternative uses d pairwise distinct symbols $\$_i$ as separators, one for each T_i, such that $\$_i < \$_j$ if and only if $i < j$. The second alternative uses the same symbol $\$$ as separator, such that $\$$ is smaller than any other symbol.

Although both approaches are straightforward and have been used in different applications [Ohlebusch 2013], they have the following drawbacks. The first alternative increases the alphabet size of T^{cat} by the number of strings, which may deteriorate the theoretical bounds of many algorithms. For instance, the workspace of [Nong 2013] would increase to $O((\sigma + d) \log N)$ bits, which is not optimal for a constant size input alphabet. The second alternative does not guarantee, for strings T_i and T_j, $i < j$, that equal suffixes of T_i and T_j will be sorted with respect to i and j, in other words, ties will not be broken by the string rank, what may cause unnecessary comparisons during suffix sorting, depending on the order in which the strings are concatenated. Moreover, this alternative may cause standard algorithms to incorrectly compute the LCP array, because LCP-values may exceed the length of the strings.

The results introduced by the thesis [Louza 2017] contribute with theoretical improvements and practical advances in building the suffix array and the BWT augmented with the LCP array, allowing the construction of full-text indexes for strings in optimal theoretical bounds, as summarized in the following sections.

2. Contributions

In the thesis we presented new algorithms to construct the suffix array and the BWT augmented with the LCP array. We also presented algorithms to construct augmented suffix arrays for string collections. The source code of our algorithms and detailed experimental results are publicly available at `https://github.com/felipelouza/`.

2.1. BWT and LCP construction in constant space

We presented an algorithm that computes the BWT in-place together with the LCP array in quadratic time using constant workspace [Louza and Telles 2015]. We also introduced the first algorithm to compute the LCP array directly in a compressed representation [Louza et al. 2017a].

We extended the elegant in-place BWT algorithm proposed by Crochemore *et al.* [Crochemore et al. 2015]. Our extension is twofold: we first show how to compute simultaneously the LCP array as well as the BWT, using constant workspace for unbounded alphabets; we then show how to build the LCP array directly in compressed representation using Elias coding [Navarro 2016], still using constant workspace and with no asymptotic slowdown. Furthermore, we provide a time/space tradeoff for our algorithm when additional memory is allowed. Our algorithm runs in quadratic time, as does Crochemore et al.'s, and is supported by interesting properties of the BWT and of the LCP array, contributing to our understanding of the time/space tradeoff curve for building indexing structures.

As future work, one can investigate whether our algorithm can be further modified to compute directly the permuted LCP array, which can be compressed in only $2n + o(n)$ bits, in quadratic or even subquadratic time.

2.2. Optimal suffix sorting and LCP construction

We introduced an algorithm to construct the suffix and LCP arrays together in linear time using $O(\sigma \log n)$ bits of workspace [Louza et al. 2017c], which is the first optimal algorithm for strings from constant alphabets of size $\sigma = O(1)$.

Our algorithm computes the LCP array combining techniques from Fischer [Fischer 2011] and Nong [Nong 2013], with a more careful usage of the available memory. Overall, our result is an improvement for the simultaneous construction of suffix and LCP arrays, achieving optimal time and space for constant alphabets. We evaluated the overhead added by computing the LCP array and showed that our algorithm is competitive in practice.

As future work one can investigate whether the recent linear non-recursive suffix array construction algorithm by Baier [Baier 2016] can also be adapted to compute the LCP array during suffix sorting.

2.3. Inducing suffix arrays for string collections

We presented algorithms to build suffix arrays for string collections [Louza et al. 2016], and showed how to compute the LCP array during suffix sorting [Louza et al. 2017b]. A highlight of our contribution is the introduction of the first optimal suffix sorting algorithms for string collections from constant alphabets.

We showed how to modify the algorithms by Nong *et al.* [Nong et al. 2011] and by Nong [Nong 2013] to construct the suffix array for a string collection maintaining their theoretical bounds while improving their practical performance. In particular, we presented an algorithm that runs in $O(N)$ time using only $O(\sigma \log N)$ bits of workspace for a collection of total length N, which is optimal for strings from alphabets of constant size $\sigma = O(1)$. Moreover, we showed how to compute the LCP array during suffix sorting with no asymptotic slowdown. We performed experiments that showed that our algorithms have an outstanding practical performance. Overall, our results improve the theoretical complexity of the solution and represent practical advances in building indexes for string collections.

As future work one can modify algorithms for a single string to handle string collections (*e.g.* [Bingmann et al. 2016, Kärkkäinen et al. 2017]). Moreover, one could improve the practical performance of our algorithms by interleaving the storage of the suffix array with the LCP array, reducing the effect of random memory accesses during suffix induction.

3. Publications

The scientific results discovered during the research for the thesis were communicated in the following articles:

1. **Felipe A. Louza**; Simon Gog; Guilherme P. Telles. Inducing enhanced suffix arrays for string collections. *Theoretical Computer Science* (QUALIS A1), v. 678: 22-39, 2017.
2. **Felipe A. Louza**; Simon Gog; Guilherme P. Telles. Optimal suffix sorting and LCP array construction for constant alphabets. *Information Processing Letters* (QUALIS A2), v. 118, 30-34, 2017.

3. **Felipe A. Louza**; Travis Gagie; Guilherme P. Telles. Burrows-Wheeler transform and LCP array construction in constant space. *Journal of Discrete Algorithms* (QUALIS B1), v. 42: 14-22, 2017.
4. **Felipe A. Louza**; Simon Gog; Guilherme P. Telles. Induced suffix sorting for string collections. *In: Proc. DCC* (QUALIS B1), 43-52, 2016.
5. **Felipe A. Louza**; Guilherme P. Telles. Computing the BWT and the LCP array in constant space. *In: Proc. IWOCA* (QUALIS B2), 312-320, 2015.

Moreover, during the Doctorate the author participated in other investigations on problems related to strings whose results were published in the following articles:

6. **Felipe A. Louza**; Guilherme P. Telles; Steve Hoffmann; Cristina D. A. Ciferri. Generalized enhanced suffix array construction in external memory. *Algorithms for Molecular Biology* (QUALIS B1), 12:26, 2017.
7. William H. A. Tustumi; Simon Gog; Guilherme P. Telles; **Felipe A. Louza**. An improved algorithm for the all-pairs suffix-prefix problem. *Journal of Discrete Algorithms* (QUALIS B1), v. 37, 34-43, 2016.
8. **Felipe A. Louza**; Simon Gog; Leandro Zanotto, Guido Araujo, Guilherme P. Telles. Parallel computation for the all-pairs suffix-prefix problem. *In: Proc. SPIRE* (QUALIS B1), 122-132, 2016.

References

[Baier 2016] Baier, U. (2016). Linear-time suffix sorting - a new approach for suffix array construction. In *Proc. An. Symp. on Combinatorial Pattern Matching (CPM)*, pages 23:1–23:12.

[Bingmann et al. 2016] Bingmann, T., Fischer, J., and Osipov, V. (2016). Inducing suffix and LCP arrays in external memory. *ACM J. Experiment. Algorithmics*, 21(2):2.3:1–2.3:27.

[Burrows and Wheeler 1994] Burrows, M. and Wheeler, D. J. (1994). A block-sorting lossless data compression algorithm. Technical report, Digital SRC Research Report.

[Crochemore et al. 2015] Crochemore, M., Grossi, R., Kärkkäinen, J., and Landau, G. M. (2015). Computing the Burrows-Wheeler transform in place and in small space. *J. Discret. Algorithms*, 32:44–52.

[Fischer 2010] Fischer, J. (2010). Wee LCP. *Inf. Process. Lett.*, 110(8-9):317–320.

[Fischer 2011] Fischer, J. (2011). Inducing the LCP-Array. In *Proc. Workshop on Algorithms and Data Structures (WADS)*, pages 374–385.

[Gog and Ohlebusch 2013] Gog, S. and Ohlebusch, E. (2013). Compressed suffix trees: Efficient computation and storage of LCP-values. *ACM J. Experiment. Algorithmics*.

[Gusfield 1997] Gusfield, D. (1997). *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, New York, NY, USA.

[Kärkkäinen 2016] Kärkkäinen, J. (2016). Suffix array construction. In *Encyclopedia of Algorithms*, pages 2141–2144. Springer.

[Kärkkäinen et al. 2017] Kärkkäinen, J., Kempa, D., Puglisi, S. J., and Zhukova, B. (2017). Engineering external memory induced suffix sorting. In *Proc. Workshop on Algorithm Engineering and Experimentation (ALENEX)*, pages 98–108.

[Kärkkäinen et al. 2009] Kärkkäinen, J., Manzini, G., and Puglisi, S. J. (2009). Permuted longest-common-prefix array. In *Proc. An. Symp. on Combinatorial Pattern Matching (CPM)*, pages 181–192.

[Kasai et al. 2001] Kasai, T., Lee, G., Arimura, H., Arikawa, S., and Park, K. (2001). Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Proc. An. Symp. on Combinatorial Pattern Matching (CPM)*, pages 181–192.

[Louza 2017] Louza, F. A. (2017). *Engineering augmented suffix sorting algorithms*. PhD thesis, University of Campinas, Brazil.

[Louza et al. 2017a] Louza, F. A., Gagie, T., and Telles, G. P. (2017a). Burrows-Wheeler transform and LCP array construction in constant space. *J. Discret. Algorithms*, 42:14–22.

[Louza et al. 2016] Louza, F. A., Gog, S., and Telles, G. P. (2016). Induced suffix sorting for string collections. In *Proc. IEEE Data Compression Conference (DCC)*, pages 43–52.

[Louza et al. 2017b] Louza, F. A., Gog, S., and Telles, G. P. (2017b). Inducing enhanced suffix arrays for string collections. *Theor. Comput. Sci.*, 678:22–39.

[Louza et al. 2017c] Louza, F. A., Gog, S., and Telles, G. P. (2017c). Optimal suffix sorting and LCP array construction for constant alphabets. *Inf. Process. Lett.*, 118:30–34.

[Louza and Telles 2015] Louza, F. A. and Telles, G. P. (2015). Computing the BWT and the LCP array in constant space. In *Proc. International Workshop on Combinatorial Algorithms (IWOCA)*, pages 312–320.

[Manber and Myers 1993] Manber, U. and Myers, E. W. (1993). Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948.

[Manzini 2004] Manzini, G. (2004). Two space saving tricks for linear time LCP array computation. In *Proc. Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 372–383.

[Navarro 2016] Navarro, G. (2016). *Compact Data Structures – A practical approach*. Cambridge University Press.

[Nong 2013] Nong, G. (2013). Practical linear-time O(1)-workspace suffix sorting for constant alphabets. *ACM Trans. Inform. Syst.*, 31(3):1–15.

[Nong et al. 2011] Nong, G., Zhang, S., and Chan, W. H. (2011). Two efficient algorithms for linear time suffix array construction. *IEEE Trans. Comput.*, 60(10):1471–1484.

[Ohlebusch 2013] Ohlebusch, E. (2013). *Bioinformatics Algorithms: Sequence Analysis, Genome Rearrangements, and Phylogenetic Reconstruction*. Oldenbusch Verlag.

[Okanohara and Sadakane 2009] Okanohara, D. and Sadakane, K. (2009). A linear-time Burrows-Wheeler transform using induced sorting. In *Proc. International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 90–101.

[Puglisi et al. 2007] Puglisi, S. J., Smyth, W. F., and Turpin, A. H. (2007). A taxonomy of suffix array construction algorithms. *ACM Comp. Surv.*, 39(2):1–31.

[Sadakane 2002] Sadakane, K. (2002). Succinct representations of LCP information and improvements in the compressed suffix arrays. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 225–232.