Trivial Object, Nontrivial Problems

Bill Smyth[1,2]

[1]Department of Computing & Software
McMaster University, Hamilton ON, Canada
smyth@mcmaster.ca

[2]Department of Informatics
King's College London

[3]School of Engineering & Information Technology
Murdoch University, Perth WA, Australia

Channel Islands, 14 March 2016

Outline

Outline

Abstract

In 1906 Axel Thue founded stringology (combinatorics on words) by describing an infinitely long sequence containing only three distinct letters (say, a, b, c) that contains no repetition; that is, no pair of adjacent equal substrings. Over the intervening century and a bit, thousands of papers have been written on various aspects, mathematical and computational, of this trivial mathematical object: the string (or word or text or sequence). Today more than ever does research flow – after all, DNA sequences are strings!

In this talk I discuss a collection of problem areas, easy to describe, not so easy to deal with:

- efficient (appropriate) computation of repetitions;
- the mysterious combinatorics of overlapping squares;
- efficient computation on "indeterminate" strings;
- characterizing strings by their "regularities";
- fast computation of global data structures.

Outline

Repetitions, Runs & Periodicity

Repetitions arise out of local periodicity in strings:

$$\begin{array}{ccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \boldsymbol{x} = & a & b & a & a & b & a & b & a & a & a \end{array} \quad (1)$$

has repetitions a^2, a^3, $(ab)^2$, $(ba)^2$ and $(aba)^2$.

$(ab)^2$ and $(ba)^2$ arise out of the same maximal periodicity or run: *ababa*. The other repetitions are runs without a tail!

Some Hard-Won Facts about Runs & Repetitions

Suppose $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is a string:

- There may be as many as $\Theta(n \log n)$ repetitions in $\boldsymbol{x}$ (Fibonacci string) and they can be computed in $\mathcal{O}(n \log n)$ time [Cro81, AP83, ML84].
- Let $\rho(n)$ be the maximum number of runs that can occur in any string of length n. Then [KK99] there exist universal positive constants k_1, k_2 such that $\rho(n) \leq k_1 n - k_2 \sqrt{n} \log_2 n$. Furthermore the runs in $\boldsymbol{x}$ can be computed in $\Theta(n)$ time [Mai89, KK99].
- After many contributions by many researchers (for example, [FSS03, Ryt06, PSS08, Gir08, CIT08, MKI^{+}08]), we now know [Sim10, BII^{+}14, FSHIL15] that

$$0.944575712 \cdots < \rho(n)/n < 0.9565 \cdots .$$

So what is the big problem???

There Oughta Be a Faster Simpler Way!

Runs are

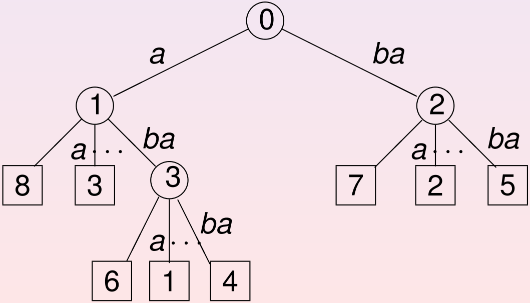
- local (independent of other segments of $\boldsymbol{x}$)
- sparse (expected number $0.4n$ in binary strings, $0.02n$ in strings on the English alphabet [PS08])
- independent of any ordering of the alphabet

but all current linear-time algorithms

- require heavy global data structures (suffix sorting)
- take no advantage of the expected sparsity of runs
- depend on an ordering of the alphabet

Suffix Trees, Suffix Arrays, et al. $\cdots$

	1	2	3	4	5	6	7	8
$\boldsymbol{x} =$	a	b	a	a	b	a	b	a
$\mathrm{SA}_{\boldsymbol{x}} =$	8	3	6	1	4	7	2	5
$\mathrm{LCP}_{\boldsymbol{x}} =$	0	1	1	3	3	0	2	2
$\mathrm{LPF}_{\boldsymbol{x}} =$	0	0	1	3	2	3	2	1
$\mathrm{BWT}_{\boldsymbol{x}} =$	b	b	b	\$	a	a	a	a



· · · then Lempel-Ziv [LZ77], finally Repetitions

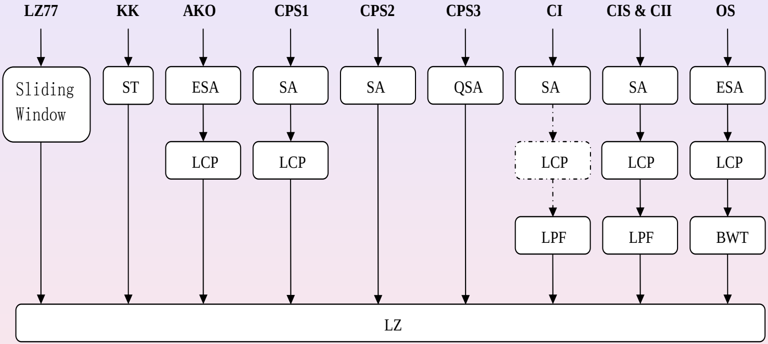


Figure : From [AHCI+13]

A Recent Ray of Light: the Lyndon Array

If $\boldsymbol{x}$ is not a repetition, it is primitive. A Lyndon word is the unique least rotation of a primitive word in some total ordering of words.

For example, in lexorder with $a \prec b$, $\boldsymbol{u} = aab$ is least among its rotations $R_0(\boldsymbol{u}) = aab, R_1(\boldsymbol{u}) = aba, R_2(\boldsymbol{u}) = baa$.

In the Lyndon array $\boldsymbol{\lambda_x} = \boldsymbol{\lambda_x}[1..n]$ of a word $\boldsymbol{x} = \boldsymbol{x}[1..n]$, $\boldsymbol{\lambda_x}[i]$ is the length of the longest Lyndon word beginning at position i of $\boldsymbol{x}$.

In a remarkable recent result, [BII$^+$14] used the computation of $\lambda_{\boldsymbol{x}}$ based on opposite orderings of the alphabet to show that $\rho(n) < n$, then went on to show that $\lambda_{\boldsymbol{x}}$ could be used to compute all the runs.

More later $\cdots$

Outline

The Periodicity Lemma

If there is a "fundamental theorem" of combinatorics on words, this is it (to avoid clutter, we write $x = |\boldsymbol{x}|$):

Lemma ("Periodicity Lemma" [FW65])

Let p and q be two periods of $\boldsymbol{x}$, and let $d = \gcd(p, q)$. If $p+q \leq x+d$, then d is also a period of $\boldsymbol{x}$.

It took 30 years to begin to think about a third square:

Lemma ("Three Squares Lemma" [CR95])

Suppose $\boldsymbol{u}$ is primitive, and suppose $\boldsymbol{v} \neq \boldsymbol{u}^j$ for any $j \geq 1$. If $\boldsymbol{u}^2$ is a prefix of $\boldsymbol{v}^2$, in turn a proper prefix of $\boldsymbol{w}^2$, then $w \geq u+v$.

The Fibostring demonstrates that this result is best possible (squares ending at positions 6, 10, $16 = 6+10$, $26 = 10+16$):

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
$\boldsymbol{x} =$	a	b	a	a	b	a	b	a	a	b	a	a	b	a	b	a	a	b	a	b	a	a	b	a	a	b

The "New Periodicity Lemma"

Lemma (NPL [FPST06, Sim07, FFSS12, KS12, BS15])

Suppose that $\boldsymbol{x}$ *has prefixes* $\boldsymbol{u}^2$ *and* $\boldsymbol{v}^2$, $3u/2 < v < 2u$, *and that* $\boldsymbol{w}^2$ *occurs at position* $k+1$ *of* $\boldsymbol{x}$, *where* $v-u < w < v$, $w \neq u$, *and* $0 \leq k < v-u$. *Then for each of 14 subcases, the structure of* $\boldsymbol{x}$ *is given below:*

Table : σ is the largest alphabet size consistent with u, v, k, w; $\boldsymbol{d}$, $\boldsymbol{d_1}$ and $\boldsymbol{d_3}$ are prefixes of $\boldsymbol{x}$ with $d = \gcd(u, v, w)$, $d_1 = \gcd(u-w, v-u)$, $d_2 = \gcd(u, v-w)$, $d_3 = v \bmod d_2$.

Subcases S	Conditions	Breakdown of $\boldsymbol{x}$
1, 2, 5, 6, 8–10	$(\forall \boldsymbol{x}, \sigma = d)$	$\boldsymbol{x} = \boldsymbol{d}^{x/d}$
3, 4, 7	$(\forall \boldsymbol{x})$	$\boldsymbol{x} = \boldsymbol{d_1}^{u/d_1} \boldsymbol{d_1}^{v/d_1} \boldsymbol{d_1}^{(v-u)/d_1}$
	specified cases	$\boldsymbol{x} = \boldsymbol{d}^{x/d}$
11–14	$\sigma = d$ or $d_2 \leq 2u-v$	$\boldsymbol{x} = \boldsymbol{d}^{x/d}$
	otherwise	$\boldsymbol{x} = \left((\boldsymbol{d_3}^{d_2/d_3})^{v/d_2}\right)^2$

(For $u < v \leq 3u/2$, a simpler result holds with even more structure.)

“New Periodicity Lemma Revisited”

We call $\boldsymbol{v}^2$ a double square $\mathrm{DS}(\boldsymbol{u}, \boldsymbol{v})$ if it has proper prefix $\boldsymbol{u}^2$. We say that $\boldsymbol{u}$ is the primitive root of $\boldsymbol{w}$ if $\boldsymbol{w} = \boldsymbol{u}^e$ for some greatest integer $e \geq 1$ (for $\boldsymbol{w} = (ab)^4$, $\boldsymbol{u} = ab, e = 4$).

Lemma (NPLR [BFS16])

Consider a double square $\mathrm{DS}(\boldsymbol{u}, \boldsymbol{v})$ with $\boldsymbol{v} = \boldsymbol{u}\boldsymbol{u}'$ for some nonempty $\boldsymbol{u}'$. Suppose that $\boldsymbol{w}^2$ is a proper substring of $\boldsymbol{v}^2$. Then exactly one of the following holds:

(a) $w < u$;

(b) $u \leq w < v$ *and the primitive root of* $\boldsymbol{w}$ *is a rotation of the primitive root of* $\boldsymbol{u}'$.

NPLR applies to somewhat fewer $\boldsymbol{w}$ than NPL, but is more precise in its characterization.

Where Do We Go From Here?

As yet no algorithm makes use of these results.

But they clearly relate to the identification of runs.

Perhaps digestion is required!

Outline

Extending the Idea of a "String"

In DNA applications it can happen that a letter is not a, c, g, t, but some combination: $\{a, c\}$, $\{g, t\}$. A regular string is defined on individual letters of an alphabet Σ; an indeterminate string is defined on indeterminate letters — nonempty subsets of Σ.

We say that λ_1 matches λ_2, written $\lambda_1 \approx \lambda_2$, if $\lambda_1 \cap \lambda_2 \neq \emptyset$; thus $\{c\} \approx \{c\}$ and $\{a, c\} \approx \{c, g\}$.

The fundamental difficulty is nontransitivity of matching: possibly $\lambda_1 \approx \lambda_2 \approx \lambda_3$, but $\lambda_1 \not\approx \lambda_3$. For example,

$$\lambda_1 = \{a, c\}, \lambda_2 = \{c, g\}, \lambda_3 = \{g, t\}.$$

Main goal: establish theory [SW09b], data structures [SW08, CRSW15] and algorithms [SW09a, ARS15, ARS16] for indeterminate strings that correspond to those for regular strings.

The Prefix Array of an Indeterminate String — I

If $\boldsymbol{u}$ is a possibly empty proper prefix of $\boldsymbol{x}$ $(0 \leq u < x)$ that matches a suffix $\boldsymbol{u}'$ of $\boldsymbol{x}$, then $\boldsymbol{u}$ is said to be a border of $\boldsymbol{x}$. The border array $\beta = \beta_{\boldsymbol{x}}[1..n]$ gives in position $i \in 1..n$ the longest border of $\boldsymbol{x}[1..i]$:

$$\begin{array}{rccc} & 1 & 2 & 3 \\ \boldsymbol{x} = & \{a, b\} & \{b, c\} & c \\ \beta = & 0 & 1 & 2 \end{array}$$

For regular strings, if $\beta[i] > 0$, then $\beta[\beta[i]]$ is the second longest border of $\boldsymbol{x}[1..i]$, and so β gives all the borders of every prefix of $\boldsymbol{x}$. The border array can be easily computed in $\Theta(n)$ time and is ubiquitous in regular string algorithms.

Alas, due to the nontransitivity of matching, this is not true for indeterminate strings: to specify all the borders, a list needs to be stored at each position of β.

The Prefix Array of an Indeterminate String — II

The prefix array $\pi = \pi_{\boldsymbol{x}}[1..n]$ gives in position i the length of the longest substring beginning at i that matches a prefix of $\boldsymbol{x}$.

$$\begin{array}{rccc} & 1 & 2 & 3 \\ \boldsymbol{x} = & \{a,b\} & \{b,c\} & c \\ \pi = & 3 & 2 & 0 \end{array}$$

For regular strings, β and π are "equivalent": one can be computed from the other in linear time. But for indeterminate strings, the prefix array retains its useful properties: $\pi_{\boldsymbol{x}}$ implicitly specifies all the borders of $\boldsymbol{x}$.

An integer array $\boldsymbol{y} = \boldsymbol{y}[1..n]$ is said to be feasible if $\boldsymbol{y}[1] = n$ and for every $i \in 2..n$, $0 \leq \boldsymbol{y}[i] \leq n+1-i$.

Lemma

Every feasible array is the prefix array of some (indeterminate) string.

The Prefix Array of an Indeterminate String — III

Problem

Given a feasible array $\boldsymbol{y}$*, find a lexicographically least string* $\boldsymbol{x}$ *(regular if possible) whose prefix array* $\pi_{\boldsymbol{x}} = \boldsymbol{y}$*.*

In [CCR09] a linear-time algorithm is described that, given a feasible array $\boldsymbol{y}$, computes a lexicographically least corresponding regular string $\boldsymbol{x}$, whenever this is possible, and otherwise returns an error message.

In [BSBW14] it is shown that a lexicographically least indeterminate string whose prefix array is $\boldsymbol{y}$ has alphabet size $\sigma \leq n+\sqrt{n}$. Then in [ARS15] an $\mathcal{O}(\sigma n^2)$-time algorithm is described that computes a lexicographically least indeterminate string whose prefix array is $\boldsymbol{y}$.

Question

Can this calculation be done any quicker? Can it be done in less than $\mathcal{O}(n^2)$ *(worst case, average case) time?*

Outline

Periodicity & Quasiperiodicity

A string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is said to have ***period*** $p = n - b$ whenever it has a border of length b. Sometimes (especially when $\boldsymbol{x}$ is a repetition or near-repetition), the minimum period can be a good descriptor of $\boldsymbol{x}$:

$$\boldsymbol{x} = (ab)^m,\ (abac)^m ab; \tag{2}$$

usually not:

$$\boldsymbol{x} = abbabaa,\ abacabad, \tag{3}$$

even when there is a lot of "regularity" in $\boldsymbol{x}$.

In [AFI91] a quasiperiod q of $\boldsymbol{x}$ was introduced: the length of a border of $\boldsymbol{x}$ such that every position of $\boldsymbol{x}$ is contained in some occurrence of $\boldsymbol{q} = \boldsymbol{x}[1..q]$. Then $\boldsymbol{q}$ is called a cover of $\boldsymbol{x}$.

[LS02] showed that the cover array $\gamma_{\boldsymbol{x}}[1..n]$ of $\boldsymbol{x}$ could be computed in $\Theta(n)$ time from the border array, specifying all the covers of every prefix of $\boldsymbol{x}$. [ARS16] showed how to compute $\gamma_{\boldsymbol{x}}$ using the prefix array, and thus extended the result to indeterminate strings using $\mathcal{O}(n)$ time on average.

Seeds & k-Covers

Unfortunately, the quasiperiod doesn't help very much: $\boldsymbol{x} = (abac)^m ab$ in (2) has no cover, nor do the strings of (3).

A seed of $\boldsymbol{x}$ is a minimum cover of a superstring of $\boldsymbol{x}$ and can be computed in $\mathcal{O}(n \log n)$ time [IMP93]. Every periodic string has a seed — for example, $(abac)^m ab$ has seed *abac*. But a seed may not help very much: in (3), *abbabaa* has seed *abbaba* and the only seed of *abacabad* is itself.

These deficiencies led to the idea of a k-cover: a minimum cardinality collection of strings, each of length k, that covers a given string $\boldsymbol{x}$. For example, both the strings of (3) have a 4-cover of size 2, perhaps not very helpful. Unfortunately, computing a k-cover is NP-complete [CIMS05], though it can be approximated to within a factor k in polynomial time [IMS11].

Enhanced/Partial String Covering

An enhanced cover $\boldsymbol{u}$ of $\boldsymbol{x}$ is a border of $\boldsymbol{x}$ that, over all the borders of $\boldsymbol{x}$, covers a maximum number of positions in $\boldsymbol{x}$. The enhanced cover array $\mathrm{EC}[1..n]$ gives the enhanced cover of every nonempty prefix of $\boldsymbol{x}$. EC can be computed in $\mathcal{O}(n \log n)$ worst-case time [FIK$^+$13] and in $\mathcal{O}(n)$ expected time, both for regular and indeterminate strings [AIR$^+$16]. No help for strings such as (3), whose borders are short and scarce.

Given an integer $\alpha \in 1..n$, an α-partial cover of $\boldsymbol{x}$ is a substring of $\boldsymbol{x}$ that covers at least α positions in $\boldsymbol{x}$; the shortest α-partial cover can be computed for all α in $\mathcal{O}(n \log n)$ time [KPR$^+$15]. Similarly there are α-partial seeds [KPR$^+$14], but computation time increases.

New ideas (new regularities) are needed: both strings (3) are one letter change away from being periodic!

Outline

The Tale of the Suffix Array

Given $\boldsymbol{x} = \boldsymbol{x}[1..n]$, the suffix array $\mathrm{SA} = \mathrm{SA}_{\boldsymbol{x}}[1..n]$ is such that for every $i \in 1..n$, $\mathrm{SA}[i] = j$ iff $\boldsymbol{x}[j..n]$ is the i^{th} suffix in some global order (such as lexorder).

1990 SA invented [MM90, MM93] hopefully to supplement the suffix tree [Wei73].

1995–2002 About 15 SACAs proposed, none of them linear-time, none lightweight [PST07].

2003 Three linear-time SACAs proposed, all recursive, all slow [KA03, KS03, KSPP03].

2004 SAs can do anything STs can do! [AKO04]

2009 A fast, recursive, linear-time, lightweight SACA is discovered [NZC09], an efficient implementation is made available on-line [Mor09].

2010– SA applications multiply, in bioinformatics and elsewhere.

What about the Lyndon Array?

1983 Computing the Lyndon array of $\boldsymbol{x}$ is equivalent to computing its Lyndon brackets, mentioned in [Lot83].

2003 [SR03] describes an $\mathcal{O}(n^2)$-time algorithm to compute Lyndon brackets, [HR03] hints at an algorithm to compute the Lyndon array from the suffix array.

2014 [BII$^+$14] uses the Lyndon array to show $\rho(n) < n$ and to compute all the runs in given $\boldsymbol{x}$ in linear time.

2016 [FHI$^+$16] describes half a dozen algorithms to compute $\lambda_{\boldsymbol{x}}$, but none of them is both linear-time and "elementary".

$\lambda_{\boldsymbol{x}} = \mathrm{NSV}(ISA_{\boldsymbol{x}})$

Definition (Next Smaller Value)

Given an array $\boldsymbol{x}[1..n]$ *of ordered values,* $\mathrm{NSV} = \mathrm{NSV}_{\boldsymbol{x}}[1..n]$ *is the* ***next smaller value array*** *of* $\boldsymbol{x}$ *if and only if for every* $i \in 1..n$, $\mathrm{NSV}[i] = j$, *where*

(a) *for every* $h \in 1..j-1,\ \boldsymbol{x}[i] \leq \boldsymbol{x}[i+h]$*; and*

(b) *either* $i+j = n+1$ *or* $\boldsymbol{x}[i] > \boldsymbol{x}[i+j]$*.*

	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{x} =$	3	8	7	10	2	1	4	9	6	5
$\mathrm{NSV}_{\boldsymbol{x}} =$	4	1	2	1	1	5	4	1	1	1

procedure NSVISA($\boldsymbol{x}[1\,..\,n]$) : $\lambda_{\boldsymbol{x}}[1\,..\,n]$
Compute $SA_{\boldsymbol{x}}$ ([NZC09, PST07])
Compute $\mathrm{ISA}_{\boldsymbol{x}}$ in place [PST07]
$\lambda_{\boldsymbol{x}} \leftarrow$ NSV($\mathrm{ISA}_{\boldsymbol{x}}$) (in place) [FHI$^+$16]
Hey Presto — linear time!

BUT ...

The suffix array $\mathrm{SA}_{\boldsymbol{x}}$ is more “global”, less “elementary” than the Lyndon array $\lambda_{\boldsymbol{x}}$: SA sorts all the suffixes of the string, $\boldsymbol{\lambda}$ just computes a local property at each position i.

Why should we need to use SA to compute λ in linear time? Why isn't there a simpler (and linear-time) algorithm?

Will we find more applications for λ as we did for SA?

A. Apostolico, M. Farach, and C.S. Iliopoulos.
Optimal superprimitivity testing for strings.
Inform. Process. Lett., 39:17–20, 1991.

Anisa Al-Hafeedh, Maxime Crochemore, Lucian Ilie, Evguenia Kopylova, W. F. Smyth, German Tischler, and Munina Yusufu.
A comparison of index-based Lempel-Ziv LZ77 factorization algorithms.
ACM Computing Surveys, 45:5:1–5:17, 2013.

Ali Alatabbi, A. S. M. Sohidull Islam, M. Sohel Rahman, Jamie Simpson, and W. F. Smyth.
Enhanced covers of regular and indeterminate strings using prefix tables.
2016.
arxiv.org/pdf/1506.06793.pdf.

Mohamed I. Abouelhoda, Stefan Kurtz, and Enno Ohlebusch.
Replacing suffix trees with enhanced suffix arrays.

J. Discrete Algorithms, 2:53–86, 2004.

Alberto Apostolico and Franco P. Preparata.
Optimal off–line detection of repetitions in a string.
Theoret. Comput. Sci., 22:297–315, 1983.

Ali Alatabbi, M. Sohel Rahman, and W. F. Smyth.
Inferring an indeterminate string from a prefix graph.
J. Discrete Algorithms, 32:6–13, 2015.

Ali Alatabbi, M. Sohel Rahman, and W. F. Smyth.
Computing covers using prefix tables.
Discrete Appl. Math., 2016.
to appear.

Haoyue Bai, Frantisek Franek, and W. F. Smyth.
The new periodicity lemma revisited.
Discrete Appl. Math., 2016.
to appear.

Hideo Bannai, Tomohiro I, Shunsuke Inenaga, Yuto Nakashima, Masayuki Takeda, and Kazuya Tsuruta.

The "runs" theorem.
2014.
arXiv:1406.0263v6.

- Widmer Bland and W. F. Smyth.
Three overlapping squares: the general case characterized & applications.
Theoret. Comput. Sci., 596:23–40, 2015.

- Francine Blanchet-Sadri, Michelle Bodnar, and Benjamin De Winkle.
New bounds and extended relations between prefix arrays, border arrays, undirected graphs, and indeterminate strings.
In N. Portier and E. Mayr, editors, *Proc. 31st Symp. on Theoretical Aspects of Computer Science*, pages 162–173, 2014.

- Julien Clément, Maxime Crochemore, and Giuseppina Rindone.
Reverse engineering prefix tables.

In *Proc. 26th Symp. on Theoretical Aspects of Computer Science*, pages 289–300, 2009.

- Richard Cole, Costas S. Iliopoulos, Manal Mohamed, and W. F. Smyth.
  The complexity of the minimum *k*-cover problem.
  J. Automata, Languages & Combinatorics, 10:641–653, 2005.

- Maxime Crochemore, Lucian Ilie, and Liviu Tinta.
  Towards a solution to the “runs” conjecture.
  In P. Ferragina and G. Landau, editors, *Proc. 19th Symp. on Combinatorial Pattern Matching*, LNCS 5029, pages 290–302. Springer-Verlag, 2008.

- Maxime Crochemore and Wojciech Rytter.
  Sqares, cubes, and time-space efficient string searching.
  Algorithmica, 13(5):405–425, 1995.

- Maxime Crochemore.
  An optimal algorithm for computing the repetitions in a word.

Information Processing Letters, 12(5):244–250, 1981.

- Manolis Christodoulakis, P. J. Ryan, W. F. Smyth, and Shu Wang.
Indeterminate strings, prefix arrays and undirected graphs.
Theoretical Comput. Sci., 600:34–48, 2015.

- Frantisek Franek, Robert C. G. Fuller, Jamie Simpson, and W. F. Smyth.
More results on overlapping squares.
J. Discrete Algorithms, 17:2–8, 2012.

- Frantisek Franek, Jan Holub, A. S. M. Sohidull Islam, M. Sohel Rahman, and W. F. Smyth.
Algorithms to compute the Lyndon array.
2016.
Submitted for publication.

- Tomas Flouri, Costas S. Iliopoulos, Tomasz Kociumaka, Solon P. Pissis, Simon J. Puglisi, W.F. Smyth, and Wojciech Tyczynski.
Enhanced string covering.

Theoretical Computer Science, 506:102 – 114, 2013.

- Kangmin Fan, Simon J. Puglisi, W. F. Smyth, and Andrew Turpin.
  A new periodicity lemma.
  SIAM J. Discrete Math., 20:656–668, 2006.

- Johannes Fischer, Štēfán Holub, Tomohiro I, and Moshe Lewenstein.
  Beyond the runs theorem.
  2015.
  arxiv.org/abs/1502.04644.

- Frantisek Franek, R. J. Simpson, and W. F. Smyth.
  The maximum number of runs in a string.
  In Mirka Miller and Kunsoo Park, editors, *Proc. 14h Australasian Workshop on Combinatorial Algorithms*, pages 26–35, 2003.

- N. J. Fine and H. S. Wilf.
  Uniqueness theorems for periodic functions.
  Proc. Amer. Math. Soc., 16:109–114, 1965.

- Mathieu Giraud.
  Not so many runs in strings.
  In Carlos Martin-Vide, Friedrich Otto, and Henning Fernau, editors, *Proc. 2nd Internat. Conf. on Language & Automata Theory & Applications*, LNCS 5196, pages 232–239. Springer-Verlag, 2008.
- Christophe Hohlweg and Christophe Reutenauer.
  Lyndon words, permutations and trees.
  Theoret. Comput. Sci., 307(1):173–178, 2003.
- C.S. Iliopoulos, D.W.G. Moore, and K. Park.
  Covering a string.
  In *Proc. 4th Symp. on Combinatorial Pattern Matching*, number 684 in Lecture Notes in Comput. Sci., pages 54–62, Berlin, Germany, 1993. Springer–Verlag.
- Costas S. Iliopoulos, Manal Mohamed, and W. F. Smyth.
  New complexity results for the k-covers problem.
  Information Sciences, 181:2571–2575, 2011.

- Pang Ko and Srinivas Aluru.
  Space efficient linear time construction of suffix arrays.
  In *Proc. 14th AnnualSymp. on Combinatorial Pattern Matching*, pages 200–210, 2003.
- Roman Kolpakov and Gregory Kucherov.
  On maximal repetitions in words.
  J. Discrete Algorithms, 1:159–186, 1999.
- Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń.
  Efficient algorithms for shortest partial seeds in words.
  In *Proc. 25th Symp. on Combinatorial Pattern Matching*, LNCS 8486, pages 192–201. Springer-Verlag, 2014.
- Tomasz Kociumaka, Solon P. Pissis, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń.
  Fast algorithm for partial covers in words.
  Algorithmica, 73:217–233, 2015.
- Juha Kärkkäinen and Peter Sanders.

Simple linear work suffix array construction.
In *Proceedings of the 30th international conference on Automata, languages and programming*, ICALP'03, pages 943–955, Berlin, Heidelberg, 2003. Springer–Verlag.

- Evguenia Kopylova and W. F. Smyth.
The three squares lemma revisited.
J. Discrete Algorithms, 11:3–14, 2012.

- Dong Kyue Kim, JeongSeop Sim, Heejin Park, and Kunsoo Park.
Linear-time construction of suffix arrays.
In Ricardo Baeza-Yates, Edgar Chávez, and Maxime Crochemore, editors, *Proc. 14th AnnualSymp. on Combinatorial Pattern Matching*, pages 186–199, 2003.

- M. Lothaire.
Combinatorics on words, Encyclopedia of Mathematics and its Applications.
Addison-Wesley Publishing Co., Reading, Mass., 1983.

- Yin Li and W. F. Smyth.

Computing the cover array in linear time.
Algorithmica, 32–1, 95–106, 2002.

A. Lempel and J. Ziv.
A universal algorithm for sequential data compression.
IEEE Trans. on Information Theory, IT–23:337–343, 1977.

Michael G. Main.
Detecting leftmost maximal periodicities.
Discrete Applied Maths., 25:145–153, 1989.

Wataru Matsubara, Kazuhiko Kusano, Akira Ishino, Hideo Bannai, and Ayumi Shinohara.
New lower bounds for the maximum number of runs in a string.
In Jan Holub and Jan Zdarek, editors, *Proc. Prague Stringology Conf.*, pages 140–145, 2008.

Michael G. Main and Richard J. Lorentz.
An $O(n \log n)$ algorithm for finding all repetitions in a string.
J. Algorithms, 5:422–432, 1984.

Udi Manber and Gene Myers.
Suffix arrays: A new method for on–line string searches.
In *Proc. 1st ACM-SIAM Symp. Discrete Algs.*, pages 319–327, 1990.

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

Yuta Mori.
libdivsufsort.
2009.
http://code.google.com/p/libdivsufsort/.

Ge Nong, Sen Zhang, and Wai H. Chan.
Linear time suffix array construction using D-critical substrings.
In Gregory Kucherov and Esko Ukkonen, editors, *20th Annual Symp. on Combinatorial Pattern Matching*, volume 5577 of *Lecture Notes in Computer Science*, pages 54–67. Springer-Verlag, 2009.

Simon J. Puglisi and R. J. Simpson.
The expected number of runs in a word.
Australasian J. Combinatorics, 42:45–54, 2008.

Simon J. Puglisi, R. J. Simpson, and W. F. Smyth.
How many runs can a string contain?
Theoret. Comput. Sci., 401:165–171, 2008.

Simon J. Puglisi, W. F. Smyth, and Andrew H. Turpin.
A taxonomy of suffix array construction algorithms.
ACM Comput. Surv., 39(2):1–31, July 2007.

Wojciech Rytter.
The number of runs in a string: improved analysis of the linear upper bound.
In B. Durand and W. Thomas, editors, *Proc. 23rd Symp. on Theoretical Aspects of Computer Science*, LNCS 2884, pages 184–195. Springer-Verlag, 2006.

R. J. Simpson.
Intersecting periodic words.

Theoret. Comput. Sci., 374:58–65, 2007.

Jamie Simpson.
Modified Padovan words and the maximum number of runs in a word.
Australasian J. Combinatorics, 46:129–145, 2010.

Joe Sawada and Frank Ruskey.
Generating Lyndon brackets: an addendum to “Fast algorithms to generate necklaces, unlabeled necklaces amd irreducible polynomials over GF(2)”.
J. Algorithms, 46:21–26, 2003.

W. F. Smyth and Shu Wang.
New perspectives on the prefix array.
Proc. 15th String Processing & Inform. Retrieval Symp. (SPIRE), 5280:133–143, 2008.

W. F. Smyth and Shu Wang.
An adaptive hybrid pattern-matching algorithm on indeterminate strings.

Internat. J. Foundations of Computer Science, 20(6):985–1004, 2009.

- W. F. Smyth and Shu Wang.
  A new approach to the periodicity lemma on strings with holes.
  Theoret. Comput. Sci., 410(43):4295–4302, 2009.

- Peter Weiner.
  Linear pattern matching algorithms.
  In *Proc. 14th Symposium on Switching and Automata Theory*, pages 1–11, 1973.