Murdoch
UNIVERSITY

Accepted Manuscript

Indeterminate strings, prefix arrays & undirected graphs

Manolis Christodoulakis, P.J. Ryan, W.F. Smyth, Shu Wang

PII: S0304-3975(15)00566-6
DOI: http://dx.doi.org/10.1016/j.tcs.2015.06.056
Reference: TCS 10313

To appear in: *Theoretical Computer Science*

Received date: 1 March 2013
Revised date: 25 June 2015
Accepted date: 26 June 2015

Please cite this article in press as: M. Christodoulakis et al., Indeterminate strings, prefix arrays & undirected graphs, *Theoret. Comput. Sci.* (2015), http://dx.doi.org/10.1016/j.tcs.2015.06.056

Indeterminate Strings, Prefix Arrays & Undirected Graphs

Manolis Christodoulakis[a], P. J. Ryan[b], W. F. Smyth[b,c,1,*], Shu Wang[d]

[a] *Department of Electrical & Computer Engineering, University of Cyprus, PO Box 20537, 1678 Nicosia, Cyprus, Email:* christodoulakis.manolis@ucy.ac.cy

[b] *Algorithms Research Group, Department of Computing & Software, McMaster University, Hamilton, Ontario, Canada L8S 4K1, Email:* {ryanpj,smyth}@mcmaster.ca, *Web:* www.cas.mcmaster.ca/cas/research/algorithms.htm

[c] *School of Engineering & Information Technology, Murdoch University, 90 South Street, Murdoch WA 6150, Australia*

[d] *IBM Toronto Software Lab, 8200 Warden Avenue, Markham, Ontario, Canada L6G 1C7, Email:* wangs@ca.ibm.com

Abstract

An integer array $\boldsymbol{y} = \boldsymbol{y}[1..n]$ is said to be ***feasible*** if and only if $\boldsymbol{y}[1] = n$ and, for every $i \in 2..n$, $i \leq i+\boldsymbol{y}[i] \leq n+1$. A string is said to be ***indeterminate*** if and only if at least one of its elements is a subset of cardinality greater than one of a given alphabet Σ; otherwise it is said to be ***regular***. A feasible array $\boldsymbol{y}$ is said to be ***regular*** if and only if it is the prefix array of some regular string. We show using a graph model that every feasible array of integers is a prefix array of some (indeterminate or regular) string, and for regular strings corresponding to $\boldsymbol{y}$, we use the model to provide a lower bound on the alphabet size. We show further that there is a 1–1 correspondence between labelled simple graphs and indeterminate strings, and we show how to determine the minimum alphabet size σ of an indeterminate string $\boldsymbol{x}$ based on its ***associated graph*** $\mathcal{G}_{\boldsymbol{x}}$. Thus, in this sense, indeterminate strings are a more natural object of combinatorial interest than the strings on elements of Σ that have traditionally been studied.

Keywords: indeterminate string, regular string, prefix array, prefix table, feasible array, undirected graph, minimum alphabet size, lexicographical order.

1. Introduction

Pattern matching in strings — that is, locating all the occurrences of a given pattern in a given text — has been studied for at least half a century. A major breakthrough was the realization that preprocessing the pattern would allow the problem to be solved significantly faster. Perhaps the first form of preprocessing

*Corresponding author.

[1] The work of the third author was supported in part by a grant from the Natural Sciences & Engineering Research Council of Canada.

was proposed in the seminal paper by Morris & Pratt [MP70], which computed the ***border array*** of the pattern; that is, an array $\boldsymbol{\beta}$, of the same length as the pattern $\boldsymbol{p}$, such that $\boldsymbol{\beta}[i]$ is the length of the longest proper prefix of $\boldsymbol{p}[1..i]$ that is also a suffix.

In recent years, a generalization of the classical string pattern matching problem has been introduced, where either the pattern or the text, or both, contain *sets* of symbols at each position, as opposed to a single symbol per position in regular strings. These types of sequences are known as ***indeterminate strings*** and were first introduced in a famous paper by Fischer & Paterson [FP74], then later studied by Abrahamson [A87]. In the last ten years or so, much work has been done by Blanchet-Sadri and her associates (for example, [BSH02]) on "strings with holes" — that is, strings on an alphabet Σ augmented by a single letter, a "hole" or "wildcard", that matches all other symbols in Σ. The monograph [B08] summarizes much of the pioneering work in this area. For indeterminate strings in their full generality, the third and fourth authors of this paper have collaborated on several papers, especially in the contexts of pattern-matching [HS03, HSW06, HSW08, SW09] and extensions to periodicity [SW08, SW09a].

In the search for a preprocessing approach to speed up the pattern matching problem on indeterminate strings, it soon became clear that the border array is of limited use. For regular strings $\boldsymbol{x}$, the border array has the desirable property that any border of a border of $\boldsymbol{x}$ is also a border of $\boldsymbol{x}$ — thus $\boldsymbol{\beta}$ implicitly specifies every border of every prefix of $\boldsymbol{x}$. For indeterminate $\boldsymbol{x}$, however, due to the nontransitivity of the match operation, this is not true [SW09, SW09a]. Hence border arrays cannot be used to speed up pattern matching on indeterminate strings. However, it turns out to be possible to make use of another data structure, the ***prefix array*** $\boldsymbol{\pi}$, in which $\boldsymbol{\pi}[i]$ is the length of the longest substring beginning at position i of $\boldsymbol{x}$ that matches a prefix of $\boldsymbol{x}$.

Apparently the first algorithm for computing the prefix array occurred as a routine in the repetitions algorithm of Main & Lorentz [ML84]; see also [S03, pp. 340–347]. A slightly improved algorithm is given in [L05, Section 8.4], and two algorithms for computing a "compressed" prefix array are described in [SW08]. A comprehensive treatment of prefix array construction algorithms can be found in [BKS13]. As noted above, for regular strings the border array and the prefix array are equivalent: it is claimed in [CHL01, CHL07], and demonstrated in detail in [BKS13], that there are $\Theta(n)$-time algorithms to compute one from the other. On the other hand, as shown in [SW08], for indeterminate strings the prefix array actually allows all borders of every prefix to be specified, while the border array does not [HS03, IMMP03]. Thus the prefix array provides a more compact and more general mechanism for identifying borders, hence for describing periodicity, in indeterminate strings.

[SW08] describes an algorithm that computes the prefix array of any indeterminate string. In this paper we consider the "reverse engineering" problem of computing a string corresponding to a given "feasible" array $\boldsymbol{y}$ — that is, any array that could conceivably be a prefix array. The first reverse engineering problem was introduced in [FLRS99, FGLR02], where a linear-time algorithm

was described to compute a lexicographically least string whose border array was a given integer array — or to return the result that no such string exists. There have been many such results published since, corresponding to other data structures and other conditions; for example, [BIST03, DLL05, FS06]. In [CCR09] a linear-time algorithm is described to compute a lexicographically least regular string $\boldsymbol{x}$ corresponding to a given feasible array $\boldsymbol{y}$, or to return an error if $\boldsymbol{y}$ corresponds to no regular string.

In this paper we solve the more general reverse engineering problem for any feasible array $\boldsymbol{y}$, regardless of whether it corresponds to a regular string or not. Moreover, we establish a remarkable connection between labelled graphs and indeterminate strings. The remainder of the paper is organized as follows. Section 2 provides preliminary information and all the necessary definitions that are used throughout the paper. In Section 3 we prove the surprising result that every feasible array is in fact a prefix array of some string (regular or indeterminate); further, we characterize the minimum alphabet size of a regular string corresponding to a given prefix array in terms of the largest clique in the negative "prefix" graph $\mathcal{P}^-$. We go on to give necessary and sufficient conditions that a given prefix array is regular. Section 4 establishes the duality between strings (whether regular or indeterminate) and labelled undirected graphs; also it provides a characterization of the minimum alphabet size of an indeterminate string $\boldsymbol{x}$ in terms of the number of "independent" maximal cliques in the "associated graph" $\mathcal{G}_{\boldsymbol{x}}$. Section 5 outlines future work.

2. Preliminaries

Traditionally, a string is a sequence of letters taken from some alphabet Σ. Since we discuss "indeterminate strings" in this paper, we begin by generalizing the definition as follows:

Definition 1. *A **string** with base alphabet Σ is either empty or else a sequence of nonempty subsets of Σ. A 1-element subset of Σ is called a **regular** letter; otherwise it is **indeterminate**. Similarly, a nonempty string consisting only of regular letters is **regular**, otherwise **indeterminate**. The empty string ε is regular.*

All alphabets and all strings discussed in this paper are finite. We denote by Σ' the set of all nonempty subsets of Σ, with $\sigma = |\Sigma|$ and $\sigma' = |\Sigma'| = 2^\sigma - 1$. On a given alphabet Σ, there are altogether $(\sigma')^n$ distinct nonempty strings of length n, of which σ^n are regular.

Definition 2. *Two elements λ, μ of Σ' are said to **match** (written $\lambda \approx \mu$) if they have nonempty intersection. Two strings $\boldsymbol{x}$, $\boldsymbol{y}$ **match** ($\boldsymbol{x} \approx \boldsymbol{y}$) if they have the same length and all corresponding letters match.*

Thus two regular letters match if and only if they are equal. But note that for indeterminate letters λ, μ, ν, it may be that $\lambda \approx \mu$ and $\lambda \approx \nu$, while $\mu \not\approx \nu$: for example, $\lambda = \{1, 2\}, \mu = 1, \nu = 2$.

Definition 3. *If a string* $\boldsymbol{x}$ *can be written* $\boldsymbol{x} = \boldsymbol{u_1 v}$ *and* $\boldsymbol{x} = \boldsymbol{w u_2}$ *for nonempty strings* $\boldsymbol{v}$, $\boldsymbol{w}$, *where* $\boldsymbol{u_1} \approx \boldsymbol{u_2}$, *then* $\boldsymbol{x}$ *is said to have a* ***border*** *of length* $|\boldsymbol{u_1}| = |\boldsymbol{u_2}|$.

Note that choosing $\boldsymbol{v} = \boldsymbol{w} = \boldsymbol{x}$ yields the empty border $\boldsymbol{\varepsilon}$ of length 0.

The ***border array*** of a string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is an integer array $\boldsymbol{\beta}[1..n]$ such that $\boldsymbol{\beta}[i]$ is the length of the longest border of $\boldsymbol{x}[1..i]$, that is, the length of the longest suffix of $\boldsymbol{x}[1..i]$ that is also a prefix of $\boldsymbol{x}[1..i]$. For regular strings $\boldsymbol{x}$, the border array $\boldsymbol{\beta}$ implicitly specifies every border of every prefix of $\boldsymbol{x}$, since any border of a border of $\boldsymbol{x}$ is also a border of $\boldsymbol{x}$. For indeterminate strings, however, due to the nontransitivity of the match operation, this is not true; for example,

$$\boldsymbol{u} = a\{a, b\}b \tag{1}$$

has a border of length 2 ($a\{a, b\} \approx \{a, b\}b$), and both borders $a\{a, b\}$ and $\{a, b\}b$ have a border of length 1 ($a \approx \{a, b\}$ and $\{a, b\} \approx b$, respectively), but $\boldsymbol{u}$ has no border of length 1. It turns out that another simple data structure can be employed to compensate for these deficiencies:

Definition 4. *The* ***prefix array*** *of a string* $\boldsymbol{x} = \boldsymbol{x}[1..n]$ *is the integer array* $\boldsymbol{y} = \boldsymbol{y}[1..n]$ *such that for every* $i \in 1..n$, $\boldsymbol{y}[i]$ *is the length of the longest prefix of* $\boldsymbol{x}[i..n]$ *that matches a prefix of* $\boldsymbol{x}$. *Thus for every prefix array* $\boldsymbol{y}$, $\boldsymbol{y}[1] = n$.

For regular strings the border array and the prefix array are equivalent and there are $\Theta(n)$-time algorithms to compute one from the other [BKS13]. On the other hand, for indeterminate strings the prefix array actually allows all borders of every prefix to be specified, while the border array does not [SW08]. For instance, in the above example (1), the prefix array of $\boldsymbol{u}$ is $\boldsymbol{y} = 320$, telling us that $\boldsymbol{u}[2..3] \approx \boldsymbol{u}[1..2]$ ($\boldsymbol{u}$ has a border of length 2), hence that $\boldsymbol{u}[2] \approx \boldsymbol{u}[1]$ (prefix $\boldsymbol{u}[1..2]$ has a border of length 1) and $\boldsymbol{u}[3] \approx \boldsymbol{u}[2]$ (suffix $\boldsymbol{u}[2..3]$ has a border of length 1), but, since $\boldsymbol{y}[3] = 0$, also that $\boldsymbol{u}$ has no border of length 1.

In order to study the "reverse engineering" problem — that is, computing the string(s) that correspond to a given prefix array — it has to be established first whether a given integer array could conceivably be a prefix array of some string. To this end, another definition is helpful:

Definition 5. *An integer array* $\boldsymbol{y} = \boldsymbol{y}[1..n]$ *such that* $\boldsymbol{y}[1] = n$ *and, for every* $i \in 2..n$,

$$0 \le \boldsymbol{y}[i] \le n+1-i, \tag{2}$$

is said to be ***feasible****. A feasible array that is a prefix array of a regular string is said to be* ***regular****.*

We will often use the condition $i \le i+\boldsymbol{y}[i] \le n+1$, equivalent to (2). Note that there are $n!$ distinct feasible arrays of length n. Recalling that there are $(2^\sigma - 1)^n$ distinct strings of length n for a fixed alphabet size σ, and applying Stirling's inequality [K68, p. 479]

$$n! > \sqrt{2\pi n}(n/e)^n,$$

where $e = 2.718\cdots$ is the base of the natural logarithm, we see that (for fixed σ) the number of feasible arrays exceeds the number of strings whenever n is large enough that

$$\sqrt{2\pi n}\Big(\frac{n}{e(2^\sigma - 1)}\Big)^n > 1. \tag{3}$$

Thus for a fixed alphabet Σ there exist feasible arrays that correspond to no (indeterminate) string on Σ. We shall see however that for unconstrained n and σ, there always exist multiple strings corresponding to any given feasible array.

3. Prefix Arrays & Indeterminate Strings

In this section, we first prove that every feasible array $\boldsymbol{y}$ is a prefix array of some string (regular or indeterminate), and show how to compute a string that corresponds to $\boldsymbol{y}$ by using a graphical representation of it. We then identify the minimum alphabet size of a regular string corresponding to a given prefix array in terms of the properties of the corresponding graph. Finally, we provide necessary and sufficient conditions that a given prefix array is regular.

3.1. The Feasible Array & the Prefix Graph

We begin with an immediate consequence of Definition 4:

Remark 6. *Let $\boldsymbol{x} = \boldsymbol{x}[1..n]$ be a string. An integer array $\boldsymbol{y} = \boldsymbol{y}[1..n]$ is the prefix array of $\boldsymbol{x}$ if and only if for each position $i \in 1..n$, the following two conditions hold:*

(a) $\boldsymbol{x}\big[1..\boldsymbol{y}[i]\big] \approx \boldsymbol{x}\big[i..i + \boldsymbol{y}[i] - 1\big]$ *;*

(b) if $i + \boldsymbol{y}[i] \leq n$, *then* $\boldsymbol{x}\big[\boldsymbol{y}[i] + 1\big] \not\approx \boldsymbol{x}\big[i + \boldsymbol{y}[i]\big]$.

We now prove the main result of this section.

Lemma 7. *Every feasible array is the prefix array of some string.*

Proof. Consider an undirected graph $\mathcal{P} = (V, E)$ whose vertex set V is the set of positions $1..n$ in a given feasible array $\boldsymbol{y}$. The edge set E is defined as follows:

$$E = \{(h, k) : \text{ for all } i \in 2..n \text{ and for all } h \in 1..\boldsymbol{y}[i], k = i + h - 1\} \tag{4}$$

We then define $\boldsymbol{x}$ as follows: for each non-isolated vertex i, let $\boldsymbol{x}[i]$ be the set of edges incident with i; for each isolated vertex i, let $\boldsymbol{x}[i]$ be the singleton set containing the loop (i, i). Let $\Sigma = E \cup L$ where L is the set of loops. We claim that $\boldsymbol{y}$ is the prefix array of $\boldsymbol{x} = \boldsymbol{x}[1..n]$.

To see this, note that for an index i such that $\boldsymbol{y}[i] > 0$, Remark 6(a) is satisfied by construction. Then, by Remark 6(b), for all $\boldsymbol{y}[i] > 0$ with $i + \boldsymbol{y}[i] \leq n$,

$$\boldsymbol{x}\big[\boldsymbol{y}[i]+1\big] \not\approx \boldsymbol{x}\big[i+\boldsymbol{y}[i]\big].$$

In case $\boldsymbol{y}[i] = 0$, Remark 6(a) is satisfied vacuously. Moreover, i is isolated and thus $\boldsymbol{x}[i] = \{(i, i)\}$, which does not match $\boldsymbol{x}[1]$; consequently, Remark 6(b) is again satisfied. Therefore, $\boldsymbol{y}$ coincides with the prefix array of $\boldsymbol{x}$, which is a string over the set Σ' of subsets of Σ. □

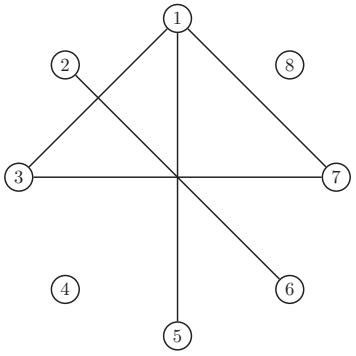


Figure 1: $\mathcal{P}^{+}_{\boldsymbol{y_1}}$ for $\boldsymbol{y_1} = 80103010$

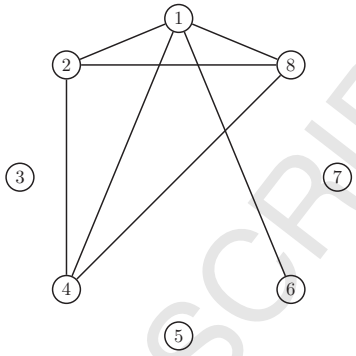


Figure 2: $\mathcal{P}^{-}_{\boldsymbol{y_1}}$ for $\boldsymbol{y_1} = 80103010$

The construction described in this proof yields a string $\boldsymbol{x}$ whose prefix array is $\boldsymbol{y}$, but $\boldsymbol{x}$ is only one string among many. For example, given the feasible array $\boldsymbol{y} = 80103010$, and applying Remark 6(a) as shown in the construction above, we get:

- $\boldsymbol{y}[3] = 1$ yields $\boldsymbol{x}[1] \approx \boldsymbol{x}[3]$
- $\boldsymbol{y}[5] = 3$ yields $\boldsymbol{x}[1] \approx \boldsymbol{x}[5]$, $\boldsymbol{x}[2] \approx \boldsymbol{x}[6]$, and $\boldsymbol{x}[3] \approx \boldsymbol{x}[7]$
- $\boldsymbol{y}[7] = 1$ yields $\boldsymbol{x}[1] \approx \boldsymbol{x}[7]$

Hence, this construction yields edges $E = \{(1,3),(1,5),(2,6),(3,7),(1,7)\}$ and loops $L = \{(4,4),(8,8)\}$, as can be seen in Fig. 1. Relabelling these seven edges/loops as a, b, c, d, e, f, g respectively, we construct $\boldsymbol{x}$ as described in the proof of Lemma 7:

$$\boldsymbol{x} = \{a,b,e\}\{c\}\{a,d\}\{f\}\{b\}\{c\}\{d,e\}\{g\}, \tag{5}$$

an indeterminate string, when in fact $\boldsymbol{y}$ is also the prefix array of the regular string $\boldsymbol{x} = abacabad$ (and so, by Definition 5, itself regular).

Definition 8. *Let $\mathcal{P} = (V, E)$ be a labelled graph with vertex set $V = \{1, 2, \ldots, n\}$ consisting of positions in a given feasible array $\boldsymbol{y}$. In $\mathcal{P}$ we define, for $i \in 2..n$, two kinds of edge (compare Remark 6):*

*(a) for every $h \in 1..\boldsymbol{y}[i]$, $(h, i+h-1)$ is called a **positive edge**;*

*(b) $(1+\boldsymbol{y}[i], i+\boldsymbol{y}[i])$ is called a **negative edge**, provided $i+\boldsymbol{y}[i] \leq n$.*

*E^+ and E^- denote the sets of positive and negative edges, respectively. We write $E = E^+ \cup E^-$, $\mathcal{P}^+ = (V, E^+)$, $\mathcal{P}^- = (V, E^-)$, and we call $\mathcal{P}$ the **prefix graph** of $\boldsymbol{y}$. If $\boldsymbol{x}$ is a string having $\boldsymbol{y}$ as its prefix array, then we also refer to $\mathcal{P}$ as the prefix graph of $\boldsymbol{x}$.*

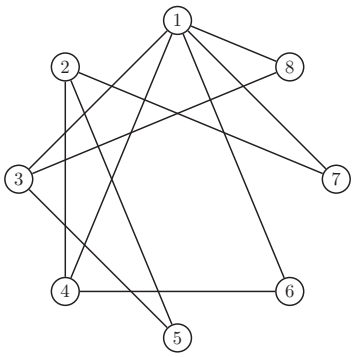


Figure 3: $\mathcal{P}^{+}_{\boldsymbol{y_2}}$ for $\boldsymbol{y_2} = 80420311$

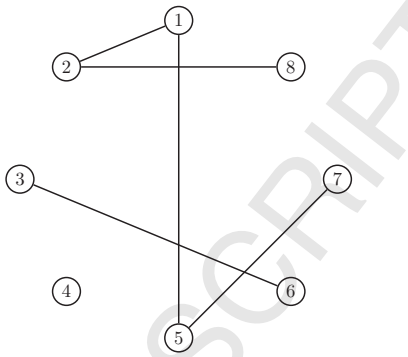


Figure 4: $\mathcal{P}^{-}_{\boldsymbol{y_2}}$ for $\boldsymbol{y_2} = 80420311$

Figures 1–4 show the prefix graphs for

	1	2	3	4	5	6	7	8
$\boldsymbol{y_1} =$	8	0	1	0	3	0	1	0
$\boldsymbol{y_2} =$	8	0	4	2	0	3	1	1

From Definition 8 it is clear that

Remark 9. *For every feasible array $\boldsymbol{y}$, there exists one and only one prefix graph $\mathcal{P}$, which therefore may be written $\mathcal{P}_{\boldsymbol{y}}$; moreover, $\mathcal{P}_{\boldsymbol{y}} = \mathcal{P}_{\boldsymbol{y'}}$ if and only if $\boldsymbol{y} = \boldsymbol{y'}$.*

Recall that a graph $\mathcal{G} = (V, E)$ is said to be ***connected*** if every pair of vertices in V is joined by a path in E. A ***connected component*** (or ***component***, for short) of $\mathcal{G}$ is a subgraph $\mathcal{G}' = (V', E')$ formed on a largest subset $V' \subseteq V$ such that every pair of vertices $i, j \in V'$ is joined by a path formed from edges $E' \subseteq E$. The graph $\mathcal{P}^{+}$ of Figure 1 has four disjoint connected components, while that of Figure 3 has only one.

The basic properties of the prefix graph $\mathcal{P}_{\boldsymbol{y}}$ of a feasible array $\boldsymbol{y} = \boldsymbol{y}[1..n]$ are as follows:

Lemma 10. *Let $\mathcal{P} = \mathcal{P}_{\boldsymbol{y}}$ be the prefix graph corresponding to a given feasible array $\boldsymbol{y}$.*

(a) E^{+} and E^{-} are disjoint and $|E^{-}| = n - s$ where s is the number of indices $i \in 1..n$ for which $i + \boldsymbol{y}[i] = n + 1$. For every $i \in 2..n$, either $(1, i) \in E^{+}$ or $(1, i) \in E^{-}$.

(b) If $(i, j) \in E^{-}$, where $i < j$, then $\boldsymbol{y}[j - i + 1] = i - 1$, and for every $h \in 1..i - 1$, $(h, j - i + h) \in E^{+}$.

(c) $\boldsymbol{y}$ is regular if and only if the end vertices of every edge of $\mathcal{P}^{-}$ occur in disjoint connected components of $\mathcal{P}^{+}$. (Thus if $\mathcal{P}^{-}$ contains an edge and $\mathcal{P}^{+}$ has only one connected component, $\boldsymbol{y}$ is not regular.)

Proof.

(a) First fix i and consider edges (h, k), where $k{-}h = i{-}1$. If $(p{+}1, p{+}i) \in E^-$ is such an edge, then the edges in E^+ must satisfy $1 \le h \le p$ and therefore are distinct from $(p{+}1, p{+}i)$. This shows that E^+ and E^- are disjoint. Secondly, $|E^-| = n{-}s$ since there is exactly one negative edge for each of the possible values of i, except those for which $i{+}\boldsymbol{y}[i] = n{+}1$. Finally, it is easily seen from Definition 8 that $(1, i)$ is a positive edge if $\boldsymbol{y}[i]$ is positive, whereas $(1, i)$ is a negative edge if $\boldsymbol{y}[i] = 0$.

(b) The first statement follows from rewriting Definition 8(b) with $j = i{+}\boldsymbol{y}[i]$, the second directly from Definition 8(a).

(c) [if] Suppose that every negative edge joins two vertices in disjoint connected components of $\mathcal{P}^+$. Form a regular string $\boldsymbol{x}$ as follows: for each component C of $\mathcal{P}^+$, assign a unique identical letter, say λ_C, to all positions $\boldsymbol{x}[i]$ for which $i \in C$. We show that $\boldsymbol{y}$ is the prefix array of $\boldsymbol{x}[1..n]$ and therefore that $\boldsymbol{y}$ is regular. Fix a value $i \in 2..n$. For any j such that $1 \le j \le \boldsymbol{y}[i]$, $(j, j{+}i{-}1)$ is a positive edge. Thus j and $j{+}i{-}1$ are in the same component of $\mathcal{P}^+$, and hence $\boldsymbol{x}[j] = \boldsymbol{x}[j + i - 1]$. We also note that $(\boldsymbol{y}[i] + 1, \boldsymbol{y}[i] + i)$ is a negative edge (provided $\boldsymbol{y}[i]{+}i \le n$). If so, then by hypothesis $\boldsymbol{y}[i] + 1$ and $\boldsymbol{y}[i] + i$ lie in disjoint components of $\mathcal{P}^+$, so that, by the uniqueness of λ_C, $\boldsymbol{x}\big[\boldsymbol{y}[i] + 1\big] \not\approx \boldsymbol{x}\big[\boldsymbol{y}[i] + i\big]$. This is precisely what we need in order to conclude that $\boldsymbol{y}$ is the prefix array of $\boldsymbol{x}[1..n]$. Since $\boldsymbol{x}$ is regular, so is $\boldsymbol{y}$, as required.

[only if] Suppose that $\boldsymbol{y}$ is regular, therefore the prefix array of a regular string $\boldsymbol{x}$. Now consider any negative edge (p, q) of the prefix graph $\mathcal{P}$ of $\boldsymbol{y}$, so that by Remark 6(b) $\boldsymbol{x}[p] \not\approx \boldsymbol{x}[q]$. If p and q were in the same component of $\mathcal{P}^+$, we would have by Remark 6(a) a path in $\mathcal{P}^+$ joining p to q consisting of edges (h, k) such that $\boldsymbol{x}[h] \approx \boldsymbol{x}[k]$. By the regularity of $\boldsymbol{y}$, this requires $\boldsymbol{x}[h] = \boldsymbol{x}[k]$, so that $\boldsymbol{x}[p] = \boldsymbol{x}[q]$, a contradiction.

□

From Definition 8, we see that $|E^+|$ can be as small as 0 (for example, when $\boldsymbol{x} = ab^{n-1}$) or as large as $\binom{n}{2}$ (when $\boldsymbol{x} = a^n$). From Lemma 10(b) we see that many of the edges in E^+ can be deduced from those in E^-. In fact, if we add an extra node $n{+}1$ and also, in the cases $i > 1$ for which $i{+}\boldsymbol{y}[i] = n{+}1$ — that is, whenever $\boldsymbol{x}$ has a border of length $\boldsymbol{y}[i] = n{+}1{-}i$ —, add the edges $(1{+}\boldsymbol{y}[i], n{+}1)$ to E^-, then all of E^+ can be deduced from E^-. Let us call this graph with the additional node and edges the ***augmented prefix graph*** and denote it by $\hat{\mathcal{P}}$ with corresponding edge sets $\hat{E}^+ = E^+$ and $\hat{E}^-$. By Lemma 10(a), $\hat{E}^-$ consists of exactly $n{-}1$ edges, which together determine $\mathcal{O}(n^2)$ edges in E^+. Of course the converse is also true: E^+ determines $\hat{E}^-$. Hence, from Remark 9, either $\mathcal{P}^+$ or $\hat{\mathcal{P}}^-$ is sufficient to determine a corresponding prefix array $\boldsymbol{y}$.

However, a bit more can be said. From Lemma 10(b) we see that every edge $(i, j) \in E^-$ determines the value $\boldsymbol{y}[j{-}i{+}1]$ of a position $j{-}i{+}1$ in $\boldsymbol{y}$. Thus a

simple scan of $\boldsymbol{y}$ can identify all positions h that are *not* determined by E^-; for all such h, it must be true that $\boldsymbol{y}[h] = n-h+1$. In other words E^- determines $\hat{E}^-$. Writing $A \equiv B$ to mean that A can be computed from B, and *vice versa*, we may summarize this discussion as follows:

Remark 11. $\boldsymbol{y} \equiv \mathcal{P}^+_{\boldsymbol{y}} \equiv \hat{\mathcal{P}}\boldsymbol{y} \equiv \hat{\mathcal{P}}^-_{\boldsymbol{y}} \equiv \mathcal{P}^-_{\boldsymbol{y}}$*: the prefix array* $\boldsymbol{y}$ *and the negative prefix graph* $\mathcal{P}^-_{\boldsymbol{y}}$ *provide the same information and so determine the same set of* (*not necessarily regular*) *strings* $\boldsymbol{x}$*; furthermore* $\mathcal{P}^-_{\boldsymbol{y}}$ *can be computed from* $\boldsymbol{y}$ *in linear time.*

In fact, we can specify a simple $\Theta(n)$-time procedure to compute $\mathcal{P}_{\boldsymbol{y}}$ from $\boldsymbol{y}$ that for each j lists in increasing order the nodes $i < j$ such that $(i, j) \in E^-$:

for $h \leftarrow 2$ **to** n **do**
  $j \leftarrow h+\boldsymbol{y}[h]$; $i \leftarrow \boldsymbol{y}[h]+1$
  if $j \leq n$ **then** $\mathcal{E}[j] \overset{+}{\leftarrow} i$

Figure 5: For each j list the edges (i, j) of E^- in increasing order of i.

Next consider Lemma 10(c). This result tells us that the regularity of $\boldsymbol{y}$ can be determined by computing the connected components of $\mathcal{P}^+_{\boldsymbol{y}}$, then determining whether or not for each edge (i, j) in $\mathcal{P}^-_{\boldsymbol{y}}$, i and j occur in different connected components of $\mathcal{P}^+_{\boldsymbol{y}}$. The algorithm formulated in [T72] computes the connected components of an undirected graph in time proportional to the number of edges; this gives rise to a straightforward algorithm to determine the regularity of $\boldsymbol{y}$ in time $\mathcal{O}(|E^+|)$. As noted below, the algorithm of [CCR09] performs this calculation in time $\mathcal{O}(|V|)$.

Recall [BM08, p. 188] that a ***t*-clique** in a graph $\mathcal{G}$ is a complete subgraph K_t of $\mathcal{G}$ on t vertices, e.g. vertices constitute 1-cliques, edges 2-cliques, triangles 3-cliques, and so on. The order t of a largest clique in $\mathcal{G}$ is called the ***clique number*** $\omega = \omega(\mathcal{G})$ of $\mathcal{G}$. Note that, $E = \emptyset \Leftrightarrow \omega = 1$, since every isolated vertex is a complete subgraph. We say that a t-clique is ***maximal*** if it is not a subclique of any $(t+1)$-clique.

Definition 12. *If* $\boldsymbol{y}$ *is a regular feasible array, then its prefix graph* $\mathcal{P}_{\boldsymbol{y}}$ *is also said to be* ***regular****.*

3.2. Lexicographically Least Regular String for a Prefix Array

We use these ideas to characterize the minimum alphabet size of any regular string with a given prefix graph $\mathcal{P}$. Suppose that the edges (i, j), $i < j$, of regular $\mathcal{P}^-$, are computed and stored according to j, as specified in Figure 5. Suppose further that, without loss of generality, $\boldsymbol{x}$ is defined on the alphabet Σ of consecutive positive integers — thus $\boldsymbol{x}$ will be lexicographically least with respect to these integers. Figure 6 describes an on-line algorithm ASSIGN that, from the lists $S[j]$ of edges in $\mathcal{P}^-$, computes a lexicographically least string $\boldsymbol{x}$ on $t = \omega(\mathcal{P}^-)$ letters whose prefix graph is $\mathcal{P}$.

```
procedure ASSIGN (P⁻, x)
    ▷ The edges (i, j) of P⁻ are available in E[j]
    ▷ in increasing order of i (Figure 5).
t ← 1; N[t] ← 0
for j ← 1 to n do
    E[j] = {(i_1, j), (i_2, j), ..., (i_r, j)}
    if r = 0 then x[j] ← 1
    ▷ Thus, if P⁻ has no edges, x = 1^n.
    else
        ▷ Determine the least letter ℓ that does not occur
        ▷ at any position i_h in E[j]; possibly ℓ = t+1.
        for h ← 1 to r do N[x[i_h]] ← 1
        ℓ ← 1
        while ℓ ≤ t and N[ℓ] = 1 do ℓ ← ℓ+1
        if ℓ > t then t ← ℓ; N[t] ← 0
        for h ← 1 to r do N[x[i_h]] ← 0
        x[j] ← ℓ
```

Figure 6: Given the negative prefix graph $\mathcal{P}^-$ of a prefix graph $\mathcal{P}$ known to be regular, compute a lexicographically least string $\boldsymbol{x}$ on $t = \omega(\mathcal{P}^-)$ letters whose prefix graph is $\mathcal{P}$.

Algorithm ASSIGN maintains a bit vector N that, for each j, specifies the letters $\boldsymbol{x}[i]$ that have occurred at positions $(i, j) \in E^-$ — that is, $N\big[\boldsymbol{x}[i]\big] = 1$. Observe that a new letter $t{+}1$ is added if and only if vertex j has an edge to vertices representing *all* previous letters $1..t$. This is true for every $t \geq 1$. Thus letter $t{+}1$ is introduced if and only if there are already t vertices that form a clique in $\mathcal{P}^-$. Consequently the number of letters used by the algorithm to form $\boldsymbol{x}$ is exactly $t = \omega(\mathcal{P}^-)$. Note also that the letter assigned at each position j is least with respect to the preceding letters, whether the letter is a new one in the string or not. Since the letters are introduced from left to right and never changed, $\boldsymbol{x}$ must therefore be lexicographically least with respect to $\mathcal{P}^-$. Note further that, since position j in the lexicographically least $\boldsymbol{x}$ is determined for $j = 1, 2, \ldots, n$ based solely on preceding positions $i < j$, it suffices to use $\mathcal{P}^-$ rather than the augmented $\hat{\mathcal{P}}^-$, in accordance with Remark 11.

Next consider the time requirement of Algorithm ASSIGN. Since we know from Lemma 10(a) that $\mathcal{P}^-$ has at most $n{-}1$ edges, it follows that, within the **for** loop, at most $n{-}1$ entries in $\mathcal{E}$ need to be accessed. The processing that updates the bit vector N, in order to determine the least letter ℓ to be assigned to $\boldsymbol{x}[j]$, requires $\Theta(r)$ time, where r is the size of $S[j]$, in order to set both $N\big[\boldsymbol{x}[i_h]\big] \leftarrow 1$ and $N\big[\boldsymbol{x}[i_h]\big] \leftarrow 0$; in addition the **while** loop requires $\mathcal{O}(r)$ time in the worst case. Since $|E^-| \leq n{-}1$, it follows that the sum of all $|S[j]| = r$ is $\mathcal{O}(n)$, and so the overall time requirement is $\Theta(n)$.

Lemma 13. *For a regular prefix graph $\mathcal{P}$ on n vertices, Algorithm ASSIGN computes in $\Theta(n)$ time a lexicographically least string on $t = \omega(\mathcal{P}^-)$ letters whose prefix graph is $\mathcal{P}$.*

Proof. We need to show that the string $\boldsymbol{x}$ computed by the algorithm is indeed consistent with $\mathcal{P}$ (that is, by Remark 11, the corresponding prefix array $\boldsymbol{y}$). Observe that S is always empty for $j = 1$, so that therefore the initial assignment $\boldsymbol{x}[1] \leftarrow 1$ is consistent with the subgraph $\mathcal{P}_1$ on a single vertex. Suppose then that $\boldsymbol{x}[1..j-1]$ has been computed by ASSIGN for some $j \in 2..n$ so as to be consistent with the subgraph $\mathcal{P}_{j-1}$ on vertices $1, 2, \ldots, j-1$. For the addition of vertex (position) j, there are three possibilities:

$|S| = 0$. In this case, $\boldsymbol{x}[j] \leftarrow 1$, the least letter, so that $\boldsymbol{x}[j] = \boldsymbol{x}[1]$, and therefore $\boldsymbol{x}[1..j]$ remains consistent with $\mathcal{P}_j^- = \mathcal{P}_{j-1}^-$.

S **gives rise to t distinct letters.** Here $\boldsymbol{x}[j] \leftarrow t{+}1$, a new letter. Since this is the first occurrence of $t{+}1$ in $\boldsymbol{x}$, and since there is no alternative, therefore $\boldsymbol{x}[1..j]$ is again consistent with $\mathcal{P}_j$ and has only the empty border.

S **gives rise to $t' < t$ distinct letters.** From the set S we know that $\boldsymbol{x}[1..j{-}1]$ has exactly r borders not continued to $\boldsymbol{x}[1..j]$. The longest of these borders is $\boldsymbol{x}[1..i_r - 1]$. There may be a border of $\boldsymbol{x}[1..j-1]$ that is on the other hand actually continued to $\boldsymbol{x}[1..j]$. If not, then the assignment $\boldsymbol{x}[j] \leftarrow \ell$ is consistent with $\mathcal{P}_j$, where ℓ is the least letter not precluded by S. Suppose then that there exists a border $\boldsymbol{x}[1..i] = \boldsymbol{x}[j-i+1..j]$, $i \geq 1$. Note that while there may be more than one such border, $\boldsymbol{x}[i]$ must be the same for each one, since we suppose that $\boldsymbol{x}$ is regular. Furthermore, $\boldsymbol{x}[i]$ was chosen by the algorithm to be a minimum letter ℓ_i with respect to the prefix $\boldsymbol{x}[1..i{-}1]$; since $\boldsymbol{x}[j{-}i{+}1..j{-}1] = \boldsymbol{x}[1..i{-}1]$, the choice of a minimum letter with respect to $\boldsymbol{x}[1..j-1]$ must yield $\ell_j = \ell_i$, hence also consistent with $\mathcal{P}_j$.

Therefore by induction the lexicographically least string $\boldsymbol{x}[1..j]$ is consistent with $\mathcal{P}_j$. We have argued above that $\boldsymbol{x}$ is lexicographically least, also that the time requirement of the algorithm is $\Theta(n)$. Thus the lemma is proved. □

Notice that the alphabet size determined by ASSIGN is least possible, given $\mathcal{P}$. Instead of assigning letters to positions in $\boldsymbol{x}$, we could just as well have labelled vertices of $\mathcal{P}$ with these letters; thus we have

Corollary 14. *The class of regular negative prefix graphs $\mathcal{P}^-$ has the property that the chromatic number $\chi(\mathcal{P}^-)$ equals the clique number $\omega(\mathcal{P}^-)$ for every graph in the class; $\chi(\mathcal{P}^-)$ is also the minimum alphabet size of the underlying string $\boldsymbol{x}$ determined by $\mathcal{P}^-$.*

This property does not hold in general; in [M55], for example, it is shown that there exist triangle-free graphs $\mathcal{G}$ ($\omega(\mathcal{G}) = 2$) with arbitrarily large chromatic number.

To get a sense of the labelling, consider the following regular prefix array

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\boldsymbol{y} =$	20	0	1	0	3	0	3	0	3	0	1	0	7	0	1	0	4	0	1	0

whose corresponding $\mathcal{P}_{\boldsymbol{y}}^-$ has edges (sorted as in Algorithm ASSIGN)

$$(1,2),\ (1,4),\ (2,4),\ (1,6),\ (1,8),\ (4,8),\ (1,10),\ (4,10),$$
$$(1,12),\ (2,12),\ (4,12),\ (1,14),\ (1,16),\ (2,16),\ (1,18),$$
$$(1,20),\ (2,20),\ (8,20).$$

$\mathcal{P}_{\boldsymbol{y}}^-$ has a single maximal clique on four vertices, $(1,2,4,12)$, and the corresponding lexicographically least string is

$$\boldsymbol{y} = abacababababadabacabac.$$

Note that $\hat{\mathcal{P}}_{\boldsymbol{y}}^-$ contains in addition the edge $(5,21)$ not required for the lexicographically least $\boldsymbol{x}$.

Now consider t-cliques $\{i_1, i_2, \ldots, i_t\}$ (not necessarily maximal) in regular prefix arrays $\mathcal{P}^-$ for which $i_1 = 1$, together with regular strings $\boldsymbol{x}$ whose prefix graph is $\mathcal{P}$. A 1-clique corresponds to a prefix $\boldsymbol{p_1} = \lambda_1$ of $\boldsymbol{x}$, where λ_1 is some (say, smallest) letter. Then for every 2-clique $(1, i_2)$ in $\mathcal{P}^-$, there must exist a corresponding prefix $\boldsymbol{p_2}$ of $\boldsymbol{x}$ such that

$$\boldsymbol{p_2} = \lambda_1 \boldsymbol{w_1} \lambda_2,$$

where $\lambda_2 > \lambda_1$ and $\boldsymbol{w_1}$ is a (possibly empty) substring. Similarly, for every 3-clique $(1, i_2, i_3)$ in $\mathcal{P}^-$, there exists a corresponding prefix $\boldsymbol{p_3}$ of $\boldsymbol{x}$ such that

$$\begin{aligned} \boldsymbol{p_3} &= \lambda_1 \boldsymbol{w_1} \lambda_2 \boldsymbol{w_2} \lambda_1 \boldsymbol{w_1} \lambda_3 \\ &= \boldsymbol{p_2} \boldsymbol{w_2} \boldsymbol{p'_2}, \end{aligned}$$

where $\boldsymbol{p_2}, \boldsymbol{p'_2}$ are identical but for distinct rightmost letters λ_2 and $\lambda_3 > \lambda_2$, respectively. In general, for every t-clique $(1, i_2, i_3, \ldots, i_t)$ in $\mathcal{P}^-$, there exists a corresponding prefix $\boldsymbol{p_t}$ of $\boldsymbol{x}$ such that

$$\boldsymbol{p_t} = \boldsymbol{p_{t-1}} \boldsymbol{w_{t-1}} \boldsymbol{p'_{t-1}},$$

where $\boldsymbol{p_{t-1}}, \boldsymbol{p'_{t-1}}$ are substrings identical but for rightmost letters λ_{t-1} and $\lambda_t > \lambda_{t-1}$, respectively. Thus every t-clique in regular $\mathcal{P}^-$ corresponds to a prefix of length $|\boldsymbol{p_t}|-1$ of the corresponding string $\boldsymbol{x}$ that has $t-2$ nonempty borders. The length of this prefix can be minimized by choosing every $\boldsymbol{w_j}$, $j \in 1..t-1$, to be empty, so that the strings $\boldsymbol{p_j}$ double in length at each step: hence there exists a prefix graph on 2^{t-1} vertices (or, equivalently, a feasible array of length 2^{t-1}) whose corresponding strings cannot be implemented on less than t letters. Thus

Lemma 15 (See also [CCR09], Proposition 4.8.)**.** *For a given regular feasible array $\boldsymbol{y} = \boldsymbol{y}[1..n]$, a regular string $\boldsymbol{x}$ whose prefix array is $\boldsymbol{y}$ can be constructed using no more than $\lfloor \log_2 n \rfloor + 1$ letters.*

[CCR09] describes a lemma more complex than Algorithm ASSIGN, but that does not require a regular prefix array as input: a nonregular feasible array is rejected at the first position detected.

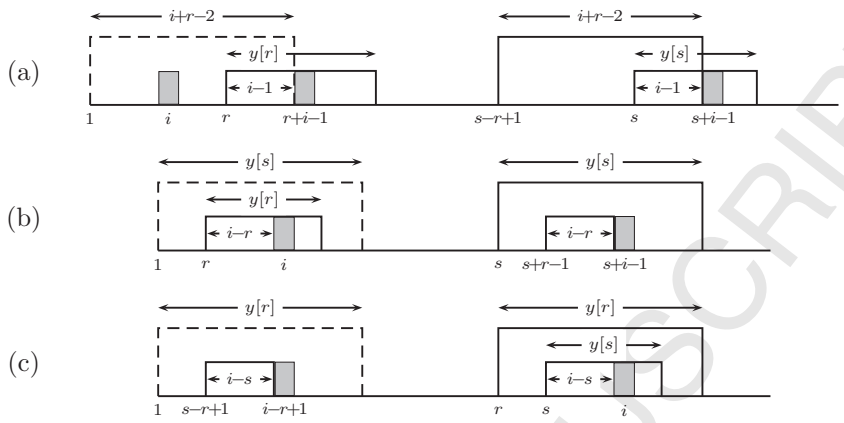


Figure 7: The three cases of Lemma 17.

3.3. Necessary & Sufficient Conditions for Regularity

We conclude this section with two equivalent necessary and sufficient conditions for $\boldsymbol{y}$ to be regular. A string $\boldsymbol{x}$ is said to be ***strongly indeterminate*** (INDET, for short) if and only if its prefix array is not regular. Recall from Definition 5 that a feasible array is regular if and only if it is a prefix array of a regular string. Thus, for example, the string (5), although certainly indeterminate, is not INDET because it is consistent with the feasible array $\boldsymbol{y} = 80103010$ that is a prefix array of the regular string $\boldsymbol{x} = abacabad$. If on the other hand $\boldsymbol{y}$ is not regular, then as we have seen (Lemma 10(c)) there must exist a position i such that $\boldsymbol{x}[i] \approx \boldsymbol{x}[r]$ and $\boldsymbol{x}[i] \approx \boldsymbol{x}[s]$, while $\boldsymbol{x}[r] \not\approx \boldsymbol{x}[s]$, for some positions r and s; in such a case we say that $\boldsymbol{x}[i]$ is ***INDET***. (In terms of the prefix graph $\mathcal{P}$, $(i,r) \in E^+,\ (i,s) \in E^+,\ (r,s) \in E^-$.)

We state two versions of what is essentially the same lemma; we prove the second.

Lemma 16. *Suppose that $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is a nonempty string with prefix array $\boldsymbol{y}$. Then for $i \in 1..n$, $\boldsymbol{x}[i]$ is INDET (and so therefore also $\boldsymbol{x}$) if and only if there exist positions r and $s > r$ such that $\boldsymbol{y}[s-r+1] = r-1$ and one of the following holds:*

(a) $\boldsymbol{y}[r-i+1] \geq i,\ \boldsymbol{y}[s-i+1] \geq i\ (1 \leq i < r < s \leq n)$;

(b) $\boldsymbol{y}[i-r+1] \geq r,\ \boldsymbol{y}[s-i+1] \geq i\ (1 \leq r < i < s \leq n)$;

(c) $\boldsymbol{y}[i-r+1] \geq r,\ \boldsymbol{y}[i-s+1] \geq s\ (1 \leq r < s < i \leq n)$.

Lemma 17. *Suppose that $\boldsymbol{x} = \boldsymbol{x}[1..n]$ is a nonempty string with prefix array $\boldsymbol{y}$. Then for $i \in 1..n$, $\boldsymbol{x}[i]$ is INDET (and so therefore also $\boldsymbol{x}$) if and only if there exist positions r and s such that one of the following holds:*

(a) $\boldsymbol{y}[r] \geq i,\ \boldsymbol{y}[s] \geq i,\ \boldsymbol{y}[s-r+1] = i+r-2$;

(b) $r+\boldsymbol{y}[r] > i,\ \boldsymbol{y}[s] \geq i,\ \boldsymbol{y}[s+r-1] = i-r;$

(c) $r+\boldsymbol{y}[r] > i,\ s+\boldsymbol{y}[s] > i,\ \boldsymbol{y}[s-r+1] = i-s.$

Proof. If $\boldsymbol{x}[i]$ is INDET, then there must exist positions r' and s' such that $\boldsymbol{x}[i] \approx \boldsymbol{x}[r'],\ \boldsymbol{x}[i] \approx \boldsymbol{x}[s'],\ \boldsymbol{x}[r'] \not\approx \boldsymbol{x}[s']$. Conversely, if such r' and s' exist, then $\boldsymbol{x}[i]$ is INDET. Without loss of generality, suppose that $s' > r'$. Then three cases arise depending on the relative values of the distinct integers i, r', s' (see Figure 7):

(a) $(1 \leq i < r' < s' \leq n)$ Since $\boldsymbol{x}[i] \approx \boldsymbol{x}[r']$ and $i < r'$, it follows that $\boldsymbol{x}[1..i] \approx \boldsymbol{x}[r'-i+1..r']$, hence that $\boldsymbol{y}[r'-i+1] \geq i$; similarly, $\boldsymbol{y}[s'-i+1] \geq i$. Since $\boldsymbol{x}[r'] \not\approx \boldsymbol{x}[s']$ and $r' < s'$, therefore $\boldsymbol{y}[s'-r'+1] = r'-1$. Setting $r \leftarrow r'-i+1,\ s \leftarrow s'-i+1$ yields the desired result.

(b) $(1 \leq r' < i < s' \leq n)$ Since $\boldsymbol{x}[i] \approx \boldsymbol{x}[r']$ and $r' < i$, therefore $\boldsymbol{x}[1..r'] \approx \boldsymbol{x}[i-r'+1..i]$, and so $\boldsymbol{y}[i-r'+1] \geq r'$; as in (a), $\boldsymbol{y}[s'-i+1] \geq i$. Also as in (a), $\boldsymbol{y}[s'-r'+1] = r'-1$. Setting $r \leftarrow i-r'+1,\ s \leftarrow s'-i+1$ yields the result.

(c) $(1 \leq r' < s' < i \leq n)$ As in (b), $\boldsymbol{y}[i-r'+1] \geq r'$; similarly, $\boldsymbol{y}[i-s'+1] \geq s'$. As in (a) and (b), $\boldsymbol{y}[s'-r'+1] = r'-1$. Setting $s \leftarrow i-r'+1,\ r \leftarrow i-s'+1$ yields the result.

□

4. Graphs & Indeterminate Strings

Here we extend the ideas of Section 3 to establish a remarkable connection between labelled graphs and indeterminate strings. Recall that a graph is ***simple*** if and only if it is undirected and contains neither loops nor multiple edges.

We define the ***associated graph***, $\mathcal{G}_{\boldsymbol{x}} = (V_{\boldsymbol{x}}, E_{\boldsymbol{x}})$, of a string $\boldsymbol{x}$ to be the simple graph whose vertices are positions $1, 2, \ldots, n$ in $\boldsymbol{x}$ and whose edges are the pairs (i, j) such that $\boldsymbol{x}[i] \approx \boldsymbol{x}[j]$. Thus $E_{\boldsymbol{x}}$ identifies *all* the matching positions in $\boldsymbol{x}$, not only those determined by the prefix array. On the other hand, we may think of each pair $(i, j) \notin E_{\boldsymbol{x}}$ as a ***negative*** edge, $\boldsymbol{x}[i] \not\approx \boldsymbol{x}[j]$. Thus $\mathcal{G}_{\boldsymbol{x}}$ determines all the pairs of positions in $\boldsymbol{x}$ that match or do not match each other.

It should be noted here that while $\mathcal{G}_{\boldsymbol{x}}$ determines the matchings of positions in $\boldsymbol{x}$, it does not uniquely determine the alphabet of $\boldsymbol{x}$. For example,

$$E_{\boldsymbol{x}} = \big\{(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(2,6),(3,5),(3,6)\big\}$$

describes

$$\boldsymbol{x_1} = \begin{array}{cccccc} {\scriptstyle 1} & {\scriptstyle 2} & {\scriptstyle 3} & {\scriptstyle 4} & {\scriptstyle 5} & {\scriptstyle 6} \\ \{a,b,c\} & \{a,b,d\} & \{a,c,d\} & b & c & d \end{array}$$

as well as

$$\boldsymbol{x_2} = \begin{array}{ccccccc} {\scriptstyle 1} & {\scriptstyle 2} & {\scriptstyle 3} & {\scriptstyle 4} & {\scriptstyle 5} & {\scriptstyle 6} \\ \{a,b\} & \{a,c\} & \{b,c\} & a & b & c \end{array}$$

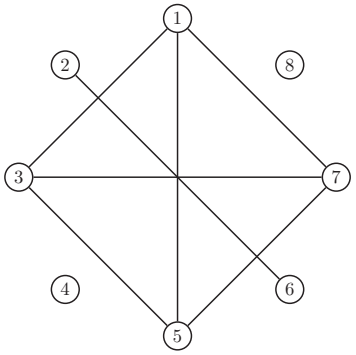


Figure 8: $\mathcal{G}_{\boldsymbol{x_3}}$ for
$\boldsymbol{x_3} = \{a,b\}\{c,d\}\{a,b\}\{e,f\}ac\{a,h\}g$

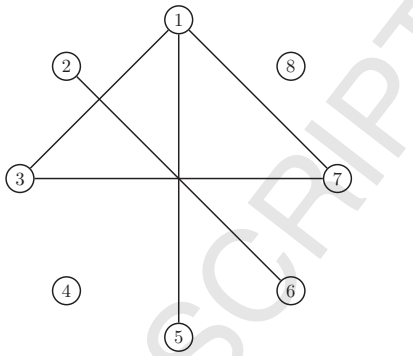
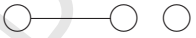
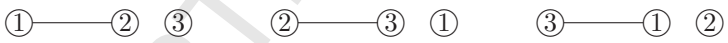


Figure 9: $\mathcal{G}_{\boldsymbol{x_4}}$ for
$\boldsymbol{x_4} = \{a,b,e\}\{c\}\{a,d\}\{f\}\{b\}\{c\}\{d,e\}\{g\}$

Thus a given simple graph $\mathcal{G} = (V, E)$ with n vertices can be the associated graph of distinct strings. Another way to generate additional strings is by permuting the vertex labels. Given any *un*labelled $\mathcal{G}$, we can generate strings $\boldsymbol{x} = \boldsymbol{x}[1..n]$ by labelling the n vertices V of $\mathcal{G}$ with integers $1..n$, and forming a string $\boldsymbol{x}$ of which $\mathcal{G}$, with this labelling, is the associated graph. Thus an unlabelled graph $\mathcal{G}$ corresponds to a set of strings $\boldsymbol{x}$ determined by the $n!$ possible labellings of V. For instance, given the graph



there are six possible labellings, three of which, for example



can be chosen to lead to distinguishable regular strings $\boldsymbol{x_1} = aab$, $\boldsymbol{x_2} = abb$, $\boldsymbol{x_3} = aba$, respectively. In this case the other three labellings determine the same three strings.

Consider a given string $\boldsymbol{x}$. Suppose that for some position $i_0 \in 1..n$, $\boldsymbol{x}[i_0]$ matches $\boldsymbol{x}[i_1], \boldsymbol{x}[i_2], \ldots, \boldsymbol{x}[i_k]$ for some $k \geq 0$, and matches no other elements of $\boldsymbol{x}$. We say that position i_0 is ***essentially regular*** if and only if the entries in positions $i_1, i_2, \ldots, i_k$ match each other pairwise. If every position in $\boldsymbol{x}$ is essentially regular, we say that $\boldsymbol{x}$ itself is ***essentially regular***. For example, the string

$$\boldsymbol{x_3} = \{a,b\}\{c,d\}\{a,b\}\{e,f\}ac\{a,h\}g,$$

with associated graph shown in Fig. 8, though indeterminate, is essentially regular with prefix array $\boldsymbol{y} = 80103010$; to see this, observe that position 1 matches positions 3, 5, and 7, which also pairwise match each other. On the other hand, string (5),

$$\boldsymbol{x_4} = \{a,b,e\}c\{a,d\}fbc\{d,e\}g,$$

also with prefix array $\boldsymbol{y}$, is not essentially regular, since position 1 again matches positions 3, 5 and 7, but position 5 does not match 3 and 7 (Fig. 9).

We have

Lemma 18. *A string $\boldsymbol{x}$ is essentially regular if and only if the associated graph $\mathcal{G}_{\boldsymbol{x}}$ of $\boldsymbol{x}$ is a disjoint union of cliques.*

Thus combinatorics on (regular, essentially regular) words is the study of labelled collections of cliques. For example, for $\boldsymbol{x} = a^n$, the associated graph $\mathcal{G}_{\boldsymbol{x}}$ is simply the complete graph K_n; while for $\boldsymbol{x}$ such that $\boldsymbol{x}[i] \approx \boldsymbol{x}[j] \Rightarrow i = j$, $\mathcal{G}_{\boldsymbol{x}}$ is n copies of K_1. More generally, for essentially regular $\boldsymbol{x}$, the number of disjoint cliques in $\mathcal{G}_{\boldsymbol{x}}$ is just the number of distinct letters in a regular string having the same associated graph as $\boldsymbol{x}$, and the order of each clique is the number of times the corresponding letter occurs.

Recall that a ***maximal clique*** (sometimes abbreviated MC) K_t in a graph $\mathcal{G} = (V, E)$ is a clique that is not a subgraph of any other clique in $\mathcal{G}$. Thus if K_t is maximal, then for every vertex j not in K_t, there exists some vertex i of K_t such that $(i, j) \notin E$. Every isolated vertex is a maximal clique K_1, and every vertex of $\mathcal{G}$ must belong to at least one maximal clique.

Definition 19. *Let $\mathcal{G} = (V, E)$ be a finite simple graph, let $\mathcal{S}$ be the set of all MC in $\mathcal{G}$, and let $\mathcal{I}$ be a smallest subset of $\mathcal{S}$ such that every vertex of V and every edge of E occur at least once in some cligue of $\mathcal{I}$. Then $\mathcal{I}$ is said to be an **independent set** in $\mathcal{G}$, and so the MC in $\mathcal{I}$ are also said to be **independent** (I), while those in $\mathcal{D} = \mathcal{S} - \mathcal{I}$ are **dependent** (D). (We see below that there may be more than one independent set in $\mathcal{G}$.)*

An edge of $\mathcal{G}$ is said to be a ***free edge*** if it belongs to exactly one MC. Then every MC that contains a free edge is necessarily an element of every independent set $\mathcal{I}$ of $\mathcal{G}$, as is every K_1. However, the converse is not true: as we discover in Figure 12, there exist graphs with no free edges.

A related idea is more useful: given an independent set $\mathcal{I} = \{I_1, I_2, \ldots, I_\sigma\}$ consisting of σ maximal cliques (IMCs for short) of a graph $\mathcal{G}$, we say that an edge e of I_j, $j \in 1..\sigma$, is a ***special edge*** if it occurs in no other IMC of $\mathcal{I}$.

Remark 20. *Every non-isolated IMC I_j, $1 \le j \le \sigma$, in an independent set $\mathcal{I} = \{I_1, I_2, \ldots, I_\sigma\}$, contains at least one special edge.*

Proof. Suppose that $\mathcal{I}$ contains a non-isolated IMC I_j with the property that every edge e in I_j also occurs in some other IMC of $\mathcal{I}$. But then I_j could be deleted from $\mathcal{I}$ without reducing the number of edges covered, contradicting the requirement in Definition 19 that $\mathcal{I}$ be "smallest". □

We will see that for the associated graph $\mathcal{G} = \mathcal{G}_{\boldsymbol{x}}$ of a string $\boldsymbol{x}$, an independent set is closely related to alphabet size. Consider for example

$$\boldsymbol{x} = \{a, b\}a\{a, c\}c\{b, c\}ab\{a, c\}, \tag{6}$$

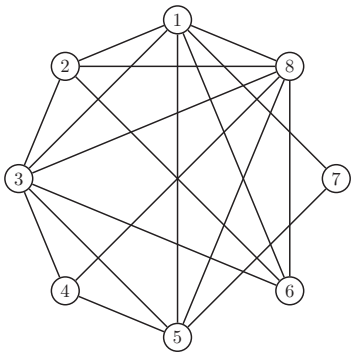


Figure 10: $\mathcal{G}_{\boldsymbol{x}}$ for
$\boldsymbol{x} = \{a,b\}a\{a,c\}c\{b,c\}ab\{a,c\}$

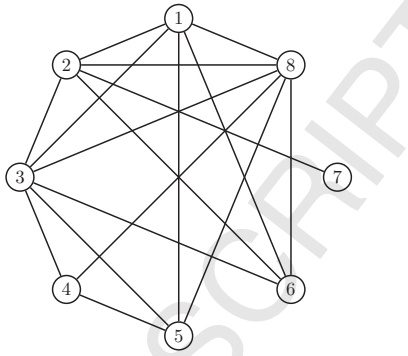


Figure 11: $\mathcal{G}_{\boldsymbol{x'}}$ for
$\boldsymbol{x'} = \{a,c\}\{a,d\}\{a,b\}b\{b,c\}ad\{a,b\}$

for which $\mathcal{G}_{\boldsymbol{x}}$ (see Figure 10) has four MC

$$C_1 = 12368,\ C_2 = 3458,\ C_3 = 1358,\ C_4 = 157, \tag{7}$$

of which, by Definition 19, C_1, C_2, C_4 are independent, since each contains at least one free edge ($(1,2), (3,4), (1,7)$, respectively). However, 1358 is dependent, since its adjacencies all occur elsewhere (138 is a subclique of C_1, 358 a subclique of C_2, 15 an edge of C_4, and so every edge of 1358 occurs in at least one of the other three cliques). Thus exactly three of the MC are independent, and we see that (6) has a minimum alphabet of three letters. On the other hand, if $\mathcal{G}_{\boldsymbol{x'}}$ (see Figure 11) has MC

$$C_1 = 12368,\ C_2 = 3458,\ C_3 = 1358,\ C_4 = 27, \tag{8}$$

all four of them are independent, and we claim that no corresponding string $\boldsymbol{x'}$ can be constructed on fewer than four letters, while

$$\boldsymbol{x'} = \overset{1}{\{a,c\}}\ \overset{2}{\{a,d\}}\ \overset{3}{\{a,b\}}\ \overset{4}{b}\ \overset{5}{\{b,c\}}\ \overset{6}{a}\ \overset{7}{d}\ \overset{8}{\{a,b\}} \tag{9}$$

achieves the lower bound.

Indeed, given an independent set $\mathcal{I} = \{I_1, I_2, \ldots, I_\sigma\}$ of a graph $\mathcal{G}$, we can define a ***canonical associated string*** $\boldsymbol{x}$, defined on exactly σ letters, as follows. Suppose that initially every $\boldsymbol{x}[i],\ i = 1, 2, \ldots, n$, is empty; then for $s = 1, 2, \ldots, \sigma$, form

$$\boldsymbol{x}[i] \leftarrow \boldsymbol{x}[i] \cup \lambda_s$$

if and only if vertex i occurs in I_s, where λ_s is a unique regular letter associated with I_s. This ensures that $\boldsymbol{x}[i_1] \approx \boldsymbol{x}[i_2]$ if and only if (i_1, i_2) is an edge in one of the IMC of $\mathcal{G}$. Since by Definition 19 this assignment includes all the vertices and all the edges of $\mathcal{G}$, it follows that $\mathcal{G} = \mathcal{G}_{\boldsymbol{x}}$ is the associated graph of $\boldsymbol{x}$, a

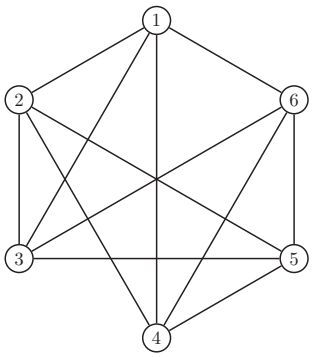


Figure 12: Graph $\mathcal{G}$ on six vertices with eight MC, four of them independent, and no free edges.

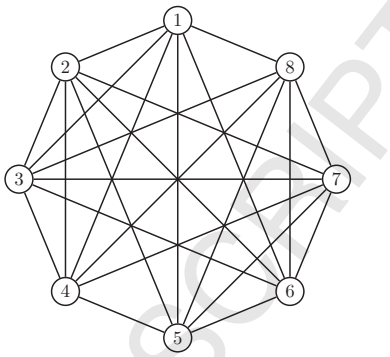


Figure 13: Graph $\mathcal{G}$ on eight vertices with 16 MC, six of them independent, and no free edges.

string on a base alphabet of size σ. For our example of Figure 11, the canonical representation is

$$\boldsymbol{x'} = \overset{1}{\{a,c\}}\ \overset{2}{\{a,d\}}\ \overset{3}{\{a,b,c\}}\ \overset{4}{b}\ \overset{5}{\{b,c\}}\ \overset{6}{a}\ \overset{7}{d}\ \overset{8}{\{a,b,c\}}, \tag{10}$$

containing two more occurrences of regular letters than (9), though with the same base alphabet.

Conjecture 21. *Suppose that $\mathcal{G} = \mathcal{G}_{\boldsymbol{x}}$ is the associated graph of $\boldsymbol{x}$ with independent set $\mathcal{I} = \{I_1, I_2, \ldots, I_\sigma\}$. Then the minimum alphabet size on which $\boldsymbol{x}$ can be built is σ.*

The following simple algorithm might be a candidate to compute an independent set:

1. Label I every MC that has a free edge;
2. Alternate steps (a) and (b) until no new labellings occur:
   (*a*) Label D each unlabelled MC with at least one edge in an MC labelled I;
   (*b*) Label I each unlabelled MC with at least one edge in an MC labelled D.

However, suppose that some subgraph $\mathcal{H}$ of $\mathcal{G}$ remains unlabelled after the termination of step 2 of the algorithm. Then every edge e of $\mathcal{H}$ must belong to at least two MC of $\mathcal{H}$, since otherwise it would have been labelled in step 1. Moreover, any MC containing e cannot be labelled either I or D, and so $\mathcal{H}$ can only be a subgraph sharing no edges with the rest of $\mathcal{G}$ and also containing no free edges.

To show that such a subgraph can exist, consider the triangulated graph $\mathcal{G}$ on six vertices $V = \{1, 2, 3, 4, 5, 6\}$, where the only pairs (i, j) that are *not* edges are $(1, 5)$, $(2, 6)$ and $(3, 4)$, as shown in Figure 12. There are eight MC

$$123, 146, 245, 356;\ 456, 124, 235, 136$$

of which either the first four or the last four can be chosen to be independent, thus by Conjecture 21 yielding a corresponding string $\boldsymbol{x}$ on four regular letters.

A more complex example is the graph $\mathcal{G}$ on vertices $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$ with maximal cliques $\{1, 2, 3, 4\}$, $\{5, 6, 7, 8\}$, and 14 others, as shown in Figure 13. The only pairs (i, j) that are *not* edges are $(1, 7)$, $(2, 8)$, $(3, 5)$, and $(4, 6)$. In this case it turns out that there are six IMC in each independent set $\mathcal{I}$, for example

$$1234, 5678, 1368, 1458, 2367, 2457,$$

and so by Conjecture 21 a corresponding string $\boldsymbol{x}$ can be constructed using six regular letters (one letter per IMC):

$$\boldsymbol{x} = \{a, c, d\}\{a, e, f\}\{a, c, e\}\{a, d, f\}\{b, d, f\}\{b, c, e\}\{b, e, f\}\{b, c, d\}.$$

These examples show that whenever graphs or subgraphs without free edges exist, the identification of independent MC becomes more difficult. In such cases we know of no algorithm to compute them apart from exhaustive search. Thus, while it is straightforward, given $\boldsymbol{x}$, to determine $\mathcal{G}_{\boldsymbol{x}}$, it is nontrivial, given $\mathcal{G}$, to determine a string $\boldsymbol{x}$ on a smallest alphabet such that $\mathcal{G} = \mathcal{G}_{\boldsymbol{x}}$.

From Lemma 18 it follows that the maximum alphabet size required for an essentially regular string $\boldsymbol{x}$ is n; thus to compute $\boldsymbol{x}$ from a feasible array $\boldsymbol{y}$ is potentially an $O(n)$ algorithm and, as shown in [CCR09], is actually $O(n)$. However, for indeterminate strings, Conjecture 21 shows that the minimum alphabet size is the number σ of independent maximal cliques in $\mathcal{G}_{\boldsymbol{x}}$. A classical result from graph theory [MM65] shows that the number of maximal cliques may be as much as $3^{n/3}$, and so an indeterminate string potentially could require an alphabet of exponential size. For example, for $n = 6$, consider the graph $\mathcal{G}_{\boldsymbol{x}}$ on six vertices $V_{\boldsymbol{x}} = \{1, 2, \ldots, 6\}$ with nine edges $(9 = 3^{6/3})$

$$E_{\boldsymbol{x}} = \{(1, 2),\ (1, 4),\ (1, 6),\ (2, 3),\ (2, 5),\ (3, 4),\ (3, 6),\ (4, 5),\ (5, 6)\},$$

as shown in Figure 14. Each of these edges is a maximal independent 2-clique, and so by Conjecture 21 a corresponding string is

$$\boldsymbol{x} = \{a, b, c\}\{a, d, e\}\{d, f, g\}\{b, f, h\}\{e, h, i\}\{c, g, i\},$$

defined on an alphabet of nine regular letters with prefix array $\boldsymbol{y} = 650301$.

Note here that information is lost in the transformation from $\boldsymbol{x}$ to $\boldsymbol{y}$. The prefix graph $\mathcal{P}^+$ corresponding to 650301 has the same nine edges $E_{\boldsymbol{x}}$, but $\mathcal{P}^-$ contains, instead of the six negative edges

$$(1, 3),\ (1, 5),\ (2, 4),\ (2, 6),\ (3, 5),\ (4, 6)$$

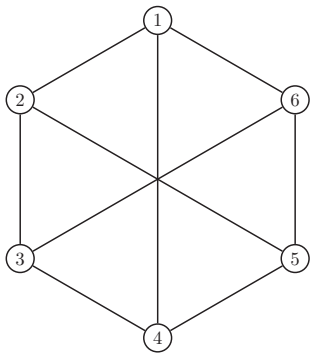


Figure 14: Identifying the minimum alphabet size from the number of independent maximal cliques (Conjecture 21)

implied by $E_{\boldsymbol{x}}$, just two: $E^- = \{(1,3),\ (1,5)\}$. Thus by reverse engineering $\boldsymbol{y}$ we get the much simpler (but still necessarily indeterminate) string

$$\boldsymbol{x'} = a\{ab\}b\{ab\}b\{ab\},$$

whose associated graph $\mathcal{G}_{\boldsymbol{x'}}$ has, in addition to the nine edges of $E_{\boldsymbol{x}}$, also the four (now positive) edges $(2,4)$, $(2,6)$, $(3,5)$, $(4,6)$. Thus in $\mathcal{G}_{\boldsymbol{x'}}$ there are only two maximal cliques, on the vertices 23456 and 1246, independent of each other, and so by Conjecture 21 $\boldsymbol{x'}$ can be constructed using $\sigma = 2$ regular letters.

The fastest known algorithm to compute all maximal cliques is described in [BK73], but of course it must be exponential in the worst case ($3^{n/3}$ maximal cliques); it is not known how many independent maximal cliques can exist in a graph constructed from a prefix array. The graph $\mathcal{P}^+$ corresponding to $\boldsymbol{y_2} = 80420311$ contains seven independent maximal cliques $(138, 146, 17, 24, 25, 27, 35)$. Thus, regarding this graph as an associated graph $\mathcal{G}_{\boldsymbol{x}}$ of some string $\boldsymbol{x}$ tells us by Conjecture 21 that seven regular letters would be needed to represent it.

5. Summary & Future Work

In this paper we have explored connections among indeterminate strings, prefix arrays, and undirected graphs, some of them quite unexpected (by us, at least). We believe that many other connections exist that may yield combinatorial insights and thus more efficient algorithms. For example:

1. How many independent maximal cliques can exist in the associated graph $\mathcal{G}_{\boldsymbol{x}}$ of a string $\boldsymbol{x}$ computed (on a minimum alphabet) from a given prefix array $\boldsymbol{y}$?
2. Find an efficient algorithm to compute a string on a minimum alphabet corresponding to a given nonregular prefix array.
3. Find an efficient algorithm to compute an associated string with a minimum number of regular letters corresponding to given graph $\mathcal{G}$.

4. What classes of graphs $\mathcal{G}$ exist that, as associated graphs $\mathcal{G} = \mathcal{G}_{\boldsymbol{x}}$ of some string $\boldsymbol{x}$, have fewer than exponential independent maximal cliques, and so therefore may give rise to efficient algorithms for the determination of $\boldsymbol{x}$ on a minimum alphabet? Put another way: characterize graphs that have an exponential number of independent maximal cliques.
5. Can we recognize strings $\boldsymbol{x}$ with associated graphs $\mathcal{G}_{\boldsymbol{x}}$ that have an exponential number of independent maximal cliques?
6. Can known results from graph theory be used to design efficient algorithms for computing patterns in indeterminate strings?

Acknowledgements

We are grateful to Jean-Pierre Duval and Arnaud Lefebvre of the Université de Rouen for useful discussions, also to Zsuzsa Lipták of the Università di Verona, and to the referees, for helpful comments.

References

[A87] Karl Abrahamson, **Generalized string matching**, *SIAM J. Computing 16–6* (1987) 1039–1051.

[BIST03] H. Bannai, S. Inenaga, A. Shinohara & M. Takeda, **Inferring strings from graphs and arrays**, *Mathematical Foundations of Computer Science*, Springer Lecture Notes in Computer Science LNCS 2747, B. Rovan & P. Vojtás (eds.) (2003) 208–217.

[B08] Francine Blanchet-Sadri, *Algorithmic Combinatorics on Partial Words*, Chapman & Hall/CRC (2008) 385 pp.

[BSH02] Francine Blanchet-Sadri & Robert A. Hegstrom, **Partial words and a theorem of Fine and Wilf revisited**, *Theoret. Comput. Sci. 270–1/2* (2002) 401–409.

[BKS13] Widmer Bland, Gregory Kucherov & W. F. Smyth, **Prefix table construction & conversion**, Proc. 24th Internat. Workshop on Combinatorial Algs., Springer Lecture Notes in Computer Science LNCS 8288, Thierry Lecroq & Laurent Mouchard (eds.) (2013) 41–53.

[BM08] J. A. Bondy & U. S. R. Murty, *Graph Theory*, Springer (2008) 651 pp.

[BK73] C. Bron & J. Kerbosch, **Algorithm 457: finding all cliques of an undirected graph**, *Communications of the ACM 16–9* (1973) 575–577.

[CCR09] Julien Clément, Maxime Crochemore & Giuseppina Rindone, **Reverse engineering prefix tables**, *Proc. 26th Symp. Theoretical Aspects of Computer Science*, Susanne Albers & Jean-Yves Marion (eds.) (2009) 289–300.

[CHL01] Maxime Crochemore, Christophe Hancart & Thierry Lecroq, *Algorithmique du Texte*, Vuibert (2001) 347 pp.

[CHL07] Maxime Crochemore, Christophe Hancart & Thierry Lecroq, *Algorithms on Strings*, Cambridge University Press (2007) 392 pp.

[DLL05] Jean-Pierre Duval, Thierry Lecroq & Arnaud Lefebvre, **Border array on a bounded alphabet**, *J. Automata, Languages & Combinatorics 10–1* (2005) 51–60.

[FP74] Michael J. Fischer & Michael S. Paterson, **String-matching and other products**, *Complexity of Computation, Proc. SIAM-AMS 7* (1974) 113-125.

[FLRS99] Frantisek Franek, Weilin Lu, P. J. Ryan, W. F. Smyth, Yu Sun & Lu Yang, **Verifying a border array in linear time** (preliminary version), *Proc. 10th Australasian Workshop on Combinatorial Algs.*, School of Computing, Curtin University of Technology (1999) 26–33.

[FGLR02] Frantisek Franek, Shudi Gao, Weilin Lu, P. J. Ryan, W. F. Smyth, Yu Sun & Lu Yang, **Verifying a border array in linear time**, *J. Combinatorial Maths. & Combinatorial Comput. 42* (2002) 223-236.

[FS06] Frantisek Franek & W. F. Smyth, **Reconstructing a suffix array**, *Internat. J. Foundations of Computer Science 17–6* (2006) 1281–1295.

[HS03] Jan Holub & W. F. Smyth, **Algorithms on indeterminate strings**, *Proc. 14th Australasian Workshop on Combinatorial Algs.* (2003) 36–45.

[HSW06] Jan Holub, W. F. Smyth & Shu Wang, **Hybrid pattern-matching algorithms on indeterminate strings**, *London Algorithmics and Stringology 2006*, J. Daykin, M. Mohamed & K. Steinhoefel (eds.), King's College London Series *Texts in Algorithmics* (2006) 115–133.

[HSW08] Jan Holub, W. F. Smyth & Shu Wang, **Fast pattern-matching on indeterminate strings**, *J. Discrete Algorithms 6–1* (2008) 37–50.

[IMMP03] Costas S. Iliopoulos, Manal Mohamed, Laurent Mouchard, Katerina G. Perdikuri, W. F. Smyth & Athanasios K. Tsakalidis, **String regularities with don't cares**, *Nordic J. Comput. 10–1* (2003) 40–51.

[K68] Joseph W. Kitchen Jr., *Calculus of One Variable*, Addison-Wesley (1968).

[L05] M. Lothaire, *Applied Combinatorics on Words*, Cambridge University Press (2005) 610 pp.

[ML84] Michael G. Main & Richard J. Lorentz, **An $O(n \log n)$ algorithm for finding all repetitions in a string**, *J. Algorithms 5* (1984) 422–432.

[MM65] J. W. Moon & L. Moser, **On cliques in graphs**, *Israel J. Math. 3* (1965) 23–28.

[MSM99] Dennis Moore, W. F. Smyth & Dianne Miller, **Counting distinct strings**, *Algorithmica 13–1* (1999) 1–13.

[MP70] James H. Morris & Vaughan R. Pratt, *A Linear Pattern-Matching Algorithm*, Tech. Rep. 40, University of California, Berkeley (1970).

[M55] J. Mycielski, **Sur le colorage des graphes**, *Colloq. Math. 3* (1955) 161–162.

[S03] Bill Smyth, *Computing Patterns in Strings*, Pearson Addison-Wesley (2003) 423 pp.

[SW08] W. F. Smyth & Shu Wang, **New perspectives on the prefix array**, *Proc. 15th String Processing & Inform. Retrieval Symp.*, Springer Lecture Notes in Computer Science LNCS 5280 (2008) 133–143.

[SW09a] W. F. Smyth & Shu Wang, **A new approach to the periodicity lemma on strings with holes**, *Theoret. Comput. Sci. 410–43* (2009) 4295–4302.

[SW09] W. F. Smyth & Shu Wang, **An adaptive hybrid pattern-matching algorithm on indeterminate strings**, *Internat. J. Foundations of Computer Science 20–6* (2009) 985–1004.

[T72] Robert Tarjan, **Depth-first search and linear graph algorithms**, *SIAM J. Computing 1* (1972) 146–160.