Universität Bielefeld

Technische Fakultät
Abteilung Informationstechnik
Forschungsberichte

Counting Suffix Arrays and Strings

Klaus-Bernd Schürmann Jens Stoye

Report 2005-04

Impressum: Herausgeber:
Robert Giegerich, Ralf Hofestädt, Franz Kummert,
Peter Ladkin, Ralf Möller, Helge Ritter,
Gerhard Sagerer, Jens Stoye, Ipke Wachsmuth

Technische Fakultät der Universität Bielefeld,
Abteilung Informationstechnik, Postfach 10 01 31,
33501 Bielefeld, Germany

ISSN 0946-7831

Counting Suffix Arrays and Strings

Klaus-Bernd Schürmann * Jens Stoye †

August 26, 2005

Abstract

Suffix arrays are used in various application and research areas like data compression or computational biology. In this work, our goal is to characterize the combinatorial properties of suffix arrays and their enumeration. For fixed alphabet size and string length we count the number of strings sharing the same suffix array and the number of such suffix arrays. Our methods have applications to succinct suffix arrays and build the foundation for the efficient generation of appropriate test data sets for suffix array based algorithms. We also show that summing up the strings for all suffix arrays builds a particular instance for some summation identities of Eulerian numbers.

1 Introduction

In the early 1990s, Manber and Myers [13] and Gonnet *et al.* [9] introduced the suffix array as an alternative data structure to suffix trees. Since then the application of and the research on suffix arrays advanced over the years [1, 2, 3, 7].

In bioinformatics and text mining applications suffix arrays with some further annotations are often used as an indexing structure for fast string querying [1], and also in the data compression community suffix arrays received more and more attention over the last decade. At first, this interest has arisen from the close relation with the Burrows-Wheeler-Transform [4] which is mainly based on the fact that computing the Burrows-Wheeler-Transform by block-sorting the input string is equivalent to suffix array construction.

Moreover, in the last years, the task of full-text index compression emerged after Grossi and Vitter introduced the compressed suffix array [11] that reduces the space requirements to a linear number of bits. Other compressed indices of that type are Ferragina and Manzini's FM-index [8] based

*AG Genominformatik, Technische Fakultät, Universität Bielefeld, Germany; Klaus-Bernd.Schuermann@CeBiTec.Uni-Bielefeld.DE

†AG Genominformatik, Technische Fakultät, Universität Bielefeld, Germany; stoye@TechFak.Uni-Bielefeld.DE

on the Burrows-Wheeler-Transform, a compressed suffix array based index of Sadakane [16] that does not use the text itself, and a suffix array based succinct index developed by He *et al.* [12], just to mention a few. Also lower bounds for the size of such indices are known. Demaine and López-Ortiz [6] proved a lower bound for indices providing substring search, and Miltersen [14] showed lower bounds for selection and rank indices.

All these developments on compressed indices, however, restrict themselves to certain queries. Therefore, information may get lost when compressing an original base index, like the suffix array. We believe that a profound knowledge of the algebraic and combinatorial properties of suffix arrays is essential to develop suffix array based, succinct indices preserving their original functionality.

Duval and Lefebvre [7] already characterized strings for the same suffix array. Further on, Crochemore *et al.* [5] recently showed combinatorial properties of the related Burrows-Wheeler transformation, but these properties are unassignable for suffix arrays. They rely on the fact that the Burrows-Wheeler transform is based on the order of cyclic shifts of the input sequence, whereas the suffix array is based on suffixes cut at the end of the string which destroys that nice group structure.

Most suffix array applications face strings with small, fixed alphabets like the DNA, amino acid, or ASCII alphabet. The possible suffix arrays for such strings are just a small fraction of all possible permutations. Therefore, besides discovering their combinatorial structure, our goal is to enumerate the different suffix arrays for strings over fixed size alphabets.

In Section 2 we give the basic definitions and notations concerning alphabets, strings, permutations, and suffix arrays. Strings sharing the same suffix array are counted in Section 4 and distinct suffix arrays in Section 5. Section 6 proves identities by summing up over suffix arrays and their strings, and Section 7 concludes.

2 Strings, Permutations, and Suffix Arrays – Definitions and Terminology.

The interval $[g, h] = \{z \in \mathbf{Z} \mid g \leq z \leq h\}$ denotes the set of all integers greater than or equal to g, and less than or equal to h.

Alphabet and Strings. Let Σ be a finite set of size $|\Sigma|$, the *alphabet*, and $t = t_1 t_2 \dots t_n \in \Sigma^n$ a string over Σ of length n, the *text*. $\Sigma(t) = \{c \in \Sigma \mid \exists i \in [1, |t|] : t_i = c\}$ is the subset of characters actually occurring in t and is called the *character set* of t. We usually use σ for the alphabet size but, if the strings are required to use all characters such that their character set equals the alphabet, we use k.

For $i \in [1,n]$, $t[i]$ denotes the i^{th} character of t, and for all pairs of indices (i,j), $1 \leq i \leq j \leq n$, $t[i,j] = t[i], t[i+1], \ldots, t[j]$ denotes the substring of t starting at position i and ending at position j. Substrings $t[i,n]$ ending at position n are *suffixes* of t. The starting position i of a suffix $t[i,n]$ is called its *suffix number*.

We deal with different kinds of equivalences of strings. The natural definition is that strings are *equivalent* if they are equal, and *distinct* otherwise.

In order to define the other two equivalences, we first introduce a bijective mapping m of the characters of a string t to the first $|\Sigma(t)|$ integers, $m : \Sigma(t) \longrightarrow [1, |\Sigma(t)|]$ such that $m(t) = m(t[1])m(t[2]) \ldots m(t[n])$. We call m *order-preserving* if $c_1 < c_2 \Leftrightarrow m(c_1) < m(c_2)$ for all pairs of characters $(c_1, c_2) \in \Sigma^2(t)$.

We call two strings t_1 and t_2 *order-equivalent*, if there exists an order-preserving bijection m_1 for t_1 and another such bijection m_2 for t_2 such that $m_1(t_1) = m_2(t_2)$; otherwise the strings are *order-distinct*. If there exist not necessarily order-preserving mappings m_1 and m_2 such that $m_1(t_1) = m_2(t_2)$, we call t_1 and t_2 *pattern-equivalent*; otherwise the strings are *pattern-distinct*.

Equivalent strings are also order-equivalent and order-equivalence implies pattern-equivalence. The strings `AT` and `AG`, for example, are distinct but order-equivalent, and the strings `AG` and `GA` are order-distinct but pattern-equivalent.

Permutations and Suffix Arrays. Let P be a permutation of $[1,n]$. Then $i \in [1, n-1]$ is a *permutation descent* if $P[i] > P[i+1]$. Conversely, a non-extendable ascending segment $P[i] < P[i+1] < \ldots < P[j]$ of P is called a *permutation run*, denoted by the index pair (i,j). Each permutation run of P is bordered by permutation descents, or the permutation boundaries 1 or n. Hence, the permutation runs define the permutation descents and vice versa.

The *suffix array* $sa(t)$ of t is a permutation of the suffix numbers $[1,n]$ according to the lexicographic ordering of the n suffixes of t. More precisely, a permutation P of $[1,n]$ is the suffix array for string t of length n if for all pairs of indices (i,j), $1 \leq i < j \leq n$, the suffix $t[P[i]], t[P[i]+1], \ldots, t[n]$ at position i in the permutation is lexicographically smaller than the suffix $t[P[j]], t[P[j]+1], \ldots, t[n]$ at position j in the permutation.

The *rank array* R_P, further on simply denoted by R, and sometimes called the inverse suffix array, for the permutation P, is defined as follows. For all indices $i \in [1,n]$ the rank of i is j, $R[i] = j$, if i occurs at position j in the permutation, $P[j] = i$. We extend the rank array by $R[n+1] = 0$, indicating that the empty suffix, not contained in the suffix array, is always the lexicographically smallest.

Further on, we define the R_+-array to be $R_+[i] = R[P[i]+1]$ for all

$i \in [1,n]$. We define the R_+-descents and R_+-runs of P similar to the permutation descents and permutation runs, respectively: A position $i \in [1,n-1]$ is called an R_+-*descent* if $R_+[i] > R_+[i+1]$. A non-extendable ascending segment $R[P[i]+1] < R[P[i+1]+1] < \ldots < R[P[j]+1]$, denoted by the index pair(i,j), $i < j$, is called an R_+-*run*. Moreover, the set of R_+-descents $\{i \in [1,n-1] \,|\, R[P[i]+1] > R[P[i+1]+1]\}$ is denoted by R_+-*desc*(P), or shortly *desc*(P), and the set of R_+-runs $\{(i,j) \in [1,n]^2 \,|\, i < j \wedge (i = 1 \vee i-1 \in desc(P)) \wedge (j = n \vee j \in desc(P)) \wedge \forall h \in [i,j-1] : h \notin desc(P)\}$ is denoted by R_+-*runs*(P), or shortly *runs*(P).

Further definitions. Besides the binomial coefficient $\binom{x}{y} = \frac{x!}{y!(x-y)!}$, combinatorial objects related to permutations that are important for this work are the Stirling numbers and the Eulerian numbers. Although these numbers have a venerable history, their notation is less standard. We will follow the notation of [10] where the Stirling number of the second kind $\left\{ {n \atop k} \right\}$ stands for the number of ways to partition a set of n elements into k nonempty subsets, and the Eulerian number $\left\langle {n \atop d} \right\rangle$ gives the number of permutations of $[1,n]$ having exactly d permutation descents, also defined through the recursion (i) $\left\langle {n \atop 0} \right\rangle = 1$, (ii) $\left\langle {n \atop d} \right\rangle = 0$ for $d \geq n$, and (iii) $\left\langle {n \atop d} \right\rangle = (d+1)\left\langle {n-1 \atop d} \right\rangle + (n-d)\left\langle {n-1 \atop d-1} \right\rangle$ for $0 < d < n$.

3 Characterizing strings sharing the same suffix array.

We repeat a characterization of the set of strings $sa^{-1}(P)$ sharing the same suffix array P that states that the order of consecutive suffixes in the suffix array is determined by their first character and by the order of suffixes with respect to offset one. This result was already given, without proof, by Burkhardt and Kärkkainen [3], and equivalent characterizations were proved by Duval and Lefebvre [7].

To start with, we generalize a proposition about consecutive elements in a permutation to arbitrary pairs of elements.

Lemma 3.1. *Let P be a permutation of $[1,n]$ and t a string of length n. If for all $i \in [1,n-1]$ we have that*

(a) $t[P[i]] \leq t[P[i+1]]$ *and*

(b) $t[P[i]] = t[P[i+1]] \;\Rightarrow\; R[P[i]+1] < R[P[i+1]+1]$,

then we also have that for all pairs (i,j), $1 \leq i < j \leq n$,

$$t[P[i]] = t[P[j]] \Rightarrow R[P[i]+1] < R[P[j]+1].$$

Proof. Due to (a), the sequence of characters $t[P[i]], t[P[i+1]], \ldots, t[P[j]]$ is non-decreasing. Combining this property with $t[P[i]] = t[P[j]]$ implies that $t[P[i']] = t[P[i]]$ for all $i' \in [i, j]$. Then, applying (b) to $t[P[i']] = t[P[i'+1]]$ leads us to $R[P[i']+1] < R[P[i'+1]+1]$ for all $i' \in [i, j-1]$, and finally by transitivity we get $R[P[i]+1] < R[P[j]+1]$. □

Before we can state the main result of this section, we continue with a further generalization. We extend our proposition from elements of the permutation referring to equal characters in the string to elements referring to starting positions of equal substrings.

Lemma 3.2. *Let P be a permutation of $[1, n]$ and t a string of length n. If for all i, $j \in [1, n]$ with $i < j$ we have that*

$$t[P[i]] = t[P[j]] \quad \Rightarrow \quad R[P[i]+1] < R[P[j]+1], \tag{1}$$

then we also have that for all i, $j \in [1, n]$ with $i < j$ and for all $k > 0$

$$t[P[i], P[i]+k-1] = t[P[j], P[j]+k-1] \Rightarrow R[P[i]+k] < R[P[j]+k]. \tag{2}$$

Proof by induction over k. For $k = 1$ the equation $t[P[i], P[i]+1-1] = t[P[j], P[j]+1-1]$ implies $t[P[i]] = t[P[j]]$. Applying Lemma 3.1 gives $R[P[i]+1] < R[P[j]+1]$.

We now perform the induction step starting with

$$t[P[i], P[i]+k] = t[P[j], P[j]+k],$$

which obviously implies

$$t[P[i], P[i]+k-1] = t[P[j], P[j]+k-1] \tag{3}$$

$$\text{and} \quad t[P[i]+k] = t[P[j]+k]. \tag{4}$$

Applying the induction hypothesis (2) to (3) gives $R[P[i]+k] < R[P[j]+k]$. Then we choose i' and j' such that $P[i'] = P[i]+k$ and $P[j'] = P[j]+k$, and since R is the inverse of P, we get

$$i' = R[P[i']] = R[P[i]+k] < R[P[j]+k] = R[P[j']] = j'. \tag{5}$$

Combining equation (4) with $P[i'] = P[i]+k$ and $P[j'] = P[j]+k$ gives

$$t[P[i']] = t[P[i]+k] = t[P[j]+k] = t[P[j']].$$

Since, by (5), i' is smaller than j', implication (1) is applicable and gives

$$R[P[i']+1] < R[P[j']+1].$$

Substituting $P[i']$ by $P[i]+k$ and $P[j']$ by $P[j]+k$ results in $R[P[i]+k+1] < R[P[j]+k+1]$, completing the proof. □

The crucial observation for the following sections is:

Theorem 3.3. *Let P be a permutation of $[1, n]$ and t a string of length n. P is the suffix array of t if and only if for all $i \in [1, n]$ the following two conditions hold:*

(a) $t[P[i]] \leq t[P[i+1]]$ *and*

(b) $R[P[i]+1] > R[P[i+1]+1] \Rightarrow t[P[i]] < t[P[i+1]]$.

Proof. If P is the suffix array for t, the conditions clearly hold.

The opposite direction is more intricate. If P is not the suffix array for t, then there must exist two wrongly ordered suffixes in P. Assume the positions of these two suffixes are i and j such that $i < j$ and $t[P[i], n] > t[P[j], n]$.

Negating (b) gives for all $i \in [1, n-1]$

$$t[P[i]] \geq t[P[i+1]] \Rightarrow R[P[i]+1] \leq R[P[i+1]+1],$$

and by (a) and by the fact that R as well as P are different at unequal positions, we get for all $i \in [1, n-1]$

$$t[P[i]] = t[P[i+1]] \Rightarrow R[P[i]+1] < R[P[i+1]+1].$$

We apply Lemma 3.1 and Lemma 3.2 to get for all $i, j \in [1, n]$, $i < j$,

$$t[P[i], P[i]+k-1] = t[P[j], P[j]+k-1] \Rightarrow R[P[i]+k] < R[P[j]+k]. \quad (6)$$

Let now l be the longest common prefix of $t[P[i], n]$ and $t[P[j], n]$. Then we distinguish between two cases.

(i) If $l = 0$, the suffixes differ in their first position. Since $t[P[i], n] > t[P[j], n]$, the first character $t[P[i]]$ of $t[P[i], n]$ must be greater than the first character $t[P[j]]$ of $t[P[j], n]$ which is a contradiction to (a).

(ii) If $l > 0$, the suffixes $t[P[i], n]$ and $t[P[j], n]$ share a longest common prefix of length l, that is, $t[P[i], P[i]+l-1] = t[P[j], P[j]+l-1]$. Then, implication (6) leads to $R[P[i]+l] < R[P[j]+l]$. We choose i' and j' such that $P[i'] = P[i]+l$ and $P[j'] = P[j]+l$. Since R is the inverse of P, we get $i' = R[P[i']] = R[P[i]+l] < R[P[j]+l] = R[P[j']] = j'$. Therefore, using (a), we obtain

$$t[P[i]+l] = t[P[i']] \leq t[P[j']] = t[P[j]+l]. \quad (7)$$

But the assumption was that $t[P[i], n] > t[P[j], n]$ with longest common prefix l such that $t[P[i]+l] > t[P[j]+l]$ which contradicts inequality (7).

Since both cases lead to contradictions, all suffixes represented in P must be in the correct order, hence P is the suffix array for t. □

Theorem 3.3 characterizes the strings in the preimage $sa^{-1}(P)$ of P, and it also suggests criteria to divide the strings in equivalence classes according to their suffix array that will be counted in Section 5.

4 Counting the strings per suffix array.

In this section, we count the number of strings over a fixed size alphabet all sharing the same suffix array.

For a permutation P with d R_+-descents, Bannai *et al.* [2] already showed that the number of different characters in a string t with suffix array P is at least the number of R_+-descents plus one, $|\Sigma(t)| \geq d + 1$. They also presented an algorithm to construct a unique string b_P consisting of exactly $d + 1$ characters, $|\Sigma(b_P)| = d + 1$.

W.l.o.g., we assume the character set of b_P contains the smallest natural numbers, $\Sigma(b_P) = [1, d + 1]$, and call b_P the *base string* of the suffix array P.

The algorithm suggested in [2] works as follows. It starts with the initial character $c = 1$. For each index position $i \in [1, n]$ in ascending order, the algorithm proceeds through all suffix numbers from $P[1]$ to $P[n]$ by setting $P[i]$ to c. If i is an R_+-descent, c is incremented by one to satisfy condition (2) of Theorem 3.3, such that $b_P[i] = d_i + 1$ where d_i is the number of descents in the prefix $P[1, \ldots, i]$ of the suffix array P.

Remark 4.1. Let P be a permutation with d R_+-descents, then the base string b_P has the properties

(a) $b_P[P[1]] = 1$ and $b_P[P[n]] = d + 1$,

(b) for $i \in ([1, n - 1] \setminus R_+\text{-desc}(P)) : b_P[P[i]] = b_P[P[i + 1]]$,

(c) for $i \in R_+\text{-desc}(P) : b_P[P[i]] + 1 = b_P[P[i + 1]]$.

4.1 Counting strings composed of up to σ distinct characters

Strings sharing the same suffix array P of length n can be derived from the base string for the suffix array by applying a certain sequence of rewrite-operations to the base string, after which the order of suffixes remains untouched. The modification sequence starts with the largest suffix. Increasing the first character of the largest suffix by r does not change the order of suffixes. Then, the first character of the second largest suffix can be increased by at most r without changing the order of suffixes, and so on.

Definition 4.2. *Let P be a permutation of $[1, n]$ with base string b_P. Moreover, let m be a sequence of length n of numbers from $[0, \psi]$, for some $\psi \in \mathbf{N}$. The* m-incremented sequence *$s_{P,m}$ of P is defined as*

$$s_{P,m}[i] = b_P[P[i]] + m[i] \quad \text{for all } i \in [1, n].$$

We show a relationship between the sequences sharing the same suffix array and non-decreasing sequences.

Theorem 4.3. *Let P be a permutation of $[1, n]$ with d R_+-descents and $\mathcal{S}_{P,\Sigma}$ the set of sequences over the ordered alphabet Σ, $\sigma = |\Sigma|$, with suffix array P. Moreover, let $\mathcal{M}$ be the set of non-decreasing sequences of length n over the ordered alphabet $[0, \sigma - d - 1]$.*

There exists an isomorphism between $\mathcal{S}_{P,\Sigma}$ and $\mathcal{M}$.

Proof. Let b_P be the base string for permutation P. W.l.o.g., we assume $\Sigma = [1, \sigma]$.

We show: (i) for each non-decreasing sequence $m \in \mathcal{M}$ the corresponding m-incremented string $s_{P,m}$ has the suffix array P and its character set is covered by Σ, and (ii) each other sequence o of length n, $o \notin \mathcal{M}$, produces a string $s_{P,o}$ for which P is not the suffix array or the character set of $s_{P,o}$ is not covered by Σ.

(i) Let $m \in \mathcal{M}$, such that $m[i] \leq m[i+1]$ for all $i \in [1, n-1]$. We verify the conditions of Theorem 3.3 for $s_{P,m}$:

(ia) For all $i \in [1, n-1]$, $b_P[P[i]] \leq b_P[P[i+1]]$. That implies

$$s_{P,m}[i] = b_P[P[i]] + m[i] \leq b_P[P[i+1]] + m[i+1] = s_{P,m}[i+1],$$

verifying Theorem 3.3(a).

(ib) If $R_+[i] > R_+[i+1]$ then $i \in R_+\text{-desc}(P)$. Hence, $b_P[P[i]] + 1 = b_P[P[i+1]]$ which leads to

$$\begin{aligned} s_{P,m}[i] &= b_P[P[i]] + m[i] \\ &< (b_P[P[i]] + 1) + m[i] \\ &\leq b_P[P[i+1]] + m[i+1] = s_{P,m}[i+1], \end{aligned}$$

verifying Theorem 3.3(b).

Therefore, P is the suffix array for $s_{P,m}$.

Moreover, for each character of $s_{P,m}$,

$$s_{P,m}[i] = b_P[P[i]] + m[i] \leq (d+1) + (\sigma - d - 1) = \sigma$$

and analogously $1 \leq s_{P,m}[i]$. Hence, each $m \in \mathcal{M}$ produces a sequence $s_{P,m}$ over the alphabet $[1, \sigma]$ with suffix array P.

(ii) For $o \notin \mathcal{M}$ containing a descending adjacent index pair such that $o[i] > o[i+1]$ for some $i \in [1, n-1]$, we concern ourselves with two cases:

(iia) If i is not an R_+-descent in P then $b_P[P[i]] = b_P[P[i+1]]$. Hence,

$$s_{P,o}[i] = b_P[P[i]] + o[i] > b_P[P[i+1]] + o[i+1] = s_{P,o}[i+1],$$

which contradicts Theorem 3.3(a).

(iib) If $R_+[i] > R_+[i+1]$ then $i \in R_+\text{-desc}(P)$. Thus, $b_P[P[i]] = b_P[P[i+1]]-1$ and, because of $o[i] > o[i+1]$, also $o[i] \geq o[i+1]+1$. This results in

$$\begin{aligned} s_{P,o}[i] &= b_P[P[i]] + o[i] \\ &\geq (b_P[P[i+1]] - 1) + (o[i+1] + 1) \\ &= b_P[P[i+1]] + o[i+1] \\ &= s_{P,o}[i+1], \end{aligned}$$

which contradicts Theorem 3.3(b).

Therefore, only the non-decreasing sequences m produce a string $s_{P,m}$ with suffix array P.

The non-decreasing sequences $o \notin \mathcal{M}$ for which the character set $\Sigma(o)$ is not covered by $[0, \sigma - d - 1]$ remain. For all these strings, we show $s_{P,o} \notin \mathcal{S}_{P,\Sigma}$.

At some position i of o, there exist a character greater than $\sigma - d - 1$ or smaller than 0. Since o is non-decreasing, this character appears at position n or 1. That is, $o[n] > \sigma - d - 1$ or $o[1] < 0$.

Combining $o[n] > \sigma - d - 1$ with the fact from Remark 4.1(a) that $b_P[P[n]] = d + 1$ gives

$$s_{P,o}[n] = b_P[P[n]] + o[n] > (d+1) + (\sigma - d - 1) = \sigma.$$

Using $b_P[P[1]] = 0$ for $o[1] < 0$ analogously, leads us to the same result. Thus, $s_{P,o} \notin \mathcal{S}_{P,\Sigma}$, completing the proof.

Finally, the number of sequences over σ characters with the same suffix array P is the same as the number of non-decreasing sequences over $\sigma - d$ characters. □

To count the number of non-decreasing sequences of length n over $k+1$ elements, we observe the following.

Lemma 4.4. *Let $M(n, a)$ be the number of non-decreasing sequences of length n of elements in $[0, a-1]$. For any positive integers n and a*

$$M(n, a) = \binom{n+a-1}{a-1}.$$

Proof. The non-decreasing sequences of length n on a symbols can be modeled as a sequence of two different operations. Initially, the current symbol is set to 0. Then, we apply a sequence of operations to generate non-decreasing sequences of length n. One possible operation is to write the current symbol behind the so far written symbols, and the other one is to increment the symbol by 1. To generate a non-decreasing sequence, we apply $n + a - 1$ operations, n to write down the non-decreasing sequence and $a - 1$ to increment the current symbol until $a - 1$ is reached. For this sequence of length $n + a - 1$, we have $\binom{n+a-1}{a-1}$ possibilities to choose the $a - 1$ positions of the increment operations. □

Applying this observation to Theorem 4.3, we get the number of strings sharing the same suffix array.

Theorem 4.5. *Let P be a permutation of length n with d R_+-descents and Σ an alphabet of $\sigma = |\Sigma|$ ordered symbols. The number of strings over Σ with suffix array P is $|S_{P,\Sigma}| = \binom{n+\sigma-d-1}{\sigma-d-1}$.*

Proof. The claim follows directly from the bijection shown in Theorem 4.3 and the equality $M(n, \sigma - d) = \binom{n+\sigma-d-1}{\sigma-d-1}$ given in Lemma 4.4. □

The non-decreasing sequences of length n over $[0, \sigma - d - 1]$ can simply be enumerated in-place by applying one change operation at a time, beginning with the sequence 0^n. The bijection described through Definition 4.2 suggests to apply these enumeration steps directly to the base string of a certain suffix array. In this way, we can enumerate all $|\mathcal{S}_{P,\Sigma}|$ strings over a given alphabet Σ for a certain suffix array P in optimal $O(n + |\mathcal{S}_{P,\Sigma}|)$ time, where n steps are used to construct the base string.

4.2 Counting strings composed of exactly k distinct characters.

So far, we have considered the strings over a fixed alphabet all sharing the same suffix array. Now, we characterize the subset of such strings all composed of exactly k different characters.

Theorem 4.6. *Let P be a permutation of length n with d R_+-descents. The number of strings with suffix array P composed of exactly k different characters is $\binom{n-d-1}{k-d-1}$.*

Proof. The proof works similar as for Theorem 4.5. Obviously, we have to count each non-decreasing sequence m in M for which $s_{P,m}$ consists of exactly k letters. To assure that none of the k characters $[1, k]$ is left out, it is sufficient to count all m such that $s_{P,m}[P[1]] = 0$, $s_{P,m}[P[n]] = k$, and consecutive characters are not differing by more than one. This property is realized by a sequence m, if and only if,

(a) $m[1] = 0$ and $m[n] = k - d - 1$,

(b) $m[i] = m[i+1]$ or $m[i] + 1 = m[i+1]$ if $i \notin R_+$-desc(P), and

(c) $m[i] = m[i+1]$ if $i \in R_+$-desc(P).

We again represent these kind of non-decreasing sequences as n write operations and $a - 1$ increment operations, as it has been modeled above. Here, for the placement of the $k - d - 1$ increment operations, we are restricted by the mentioned conditions.

In order not to hurt these conditions, (a) an increment operation must not appear before or after the first or last write operation, (b) at most one increment operation must appear between two write operations, and (c) the d descent positions are blocked for the increments. We are thus left with $n - 1 - d$ mutually exclusive positions from which we choose $k - d - 1$ increment operations. □

4.3 Filling the gaps.

For a given permutation P of length n with d R_+-descents, we have already counted the strings over an alphabet of size σ and the strings composed of exactly k distinct characters, respectively.

Table 1 summarizes the results. For different conditions, it shows the number of distinct, order-distinct, and pattern-distinct strings of length n. The first row shows the number of strings composed of exactly k different characters, the second row the number of strings over a certain alphabet of size σ, and the third and fourth rows the number of such strings sharing the same suffix array.

Some of the numbers were proven by other authors or in the previous sections, but there are yet some gaps to be filled. We start with the first row. Moore *et al.* [15] already showed that the number of pattern-distinct strings composed of exactly k different characters is $\left\{{n \atop k}\right\}$. For each pattern-distinct string, we permute the alphabet in $k!$ different ways to get a total of $\left\{{n \atop k}\right\}k!$ order-distinct strings. These are already all the distinct strings since we have no flexibility to choose different characters to produce distinct strings yet order-equivalent.

The numbers of strings over a given alphabet of size σ are shown in the second row. Needless to say, we have σ^n distinct strings. For the order-

number of	distinct	order-distinct	pattern-distinct
strings with exactly k letters	$\left\{ {n \atop k} \right\} \cdot k!$	$\left\{ {n \atop k} \right\} \cdot k!$	$\left\{ {n \atop k} \right\}$ [15]
strings for alphabet size σ	σ^n	$\sum_{k=1}^{\sigma} \left\{ {n \atop k} \right\} \cdot k!$	$\sum_{k=1}^{\sigma} \left\{ {n \atop k} \right\}$
string with exactly k letters sharing same suffix array	$\binom{n-d-1}{k-d-1}$ [Thm. 4.6]	$\binom{n-d-1}{k-d-1}$	–
strings for alphabet size σ sharing same suffix array	$\binom{n+\sigma-d-1}{\sigma-d-1}$ [Thm. 4.5]	$\sum_{k=d+1}^{\sigma} \binom{n-d-1}{k-d-1}$	–

Table 1: Number of distinct, order-distinct and pattern-distinct strings of length n in general, and those mapped to the same suffix array. In the analyses, d is always the number of R_+-descents for the respective suffix array.

and pattern-distinct strings, we just sum up the number of strings for all possible k.

The number of distinct strings composed of exactly k different characters sharing a suffix array P with d R_+-descents was given in Theorem 4.6. All these strings are again order-distinct. For a pattern-distinct string, we cannot necessarily determine a unique suffix array. For example, *ab* and *ba* are pattern-equivalent, but have different suffix arrays. This is indicated by a dash in the table.

The number of distinct and order-distinct strings over an alphabet of size σ sharing the same suffix array are given in the fourth row. Theorem 4.5 gave the number of distinct strings, and for the order-distinct strings we just sum up over all possible k.

5 Counting suffix arrays for strings with fixed alphabet.

In this section, the distinct suffix arrays for strings over a fixed size alphabet are counted. This also yields a tight lower bound for the compressibility of suffix arrays.

We first confine ourselves to the equivalent problem of counting the number of suffix arrays with a certain number of R_+-descents.

Bannai *et al.* [2] already stated that the number of suffix arrays of length n with exactly d R_+-descents is equal to the Eulerian number $\left\langle {n \atop d} \right\rangle$. In their explanation, they interpret Eulerian numbers as the number of permutations of length n with d permutation descents, and explain how their algorithm checks for these permutation descents. In fact, their algorithm counts the number of R_+-descents, but the R_+-array is not a permutation. Nevertheless, as we show in this section, their proposition is true.

For a permutation P of length $n-1$, we map P to a set $\mathcal{P}'$ of successor permutations, each of length n. We show some relations between P and $\mathcal{P}'$, finally leading to the recursive definition of the Eulerian numbers.

First of all, we define the mapping from P to $\mathcal{P}'$.

Definition 5.1. *Let P be a permutation of length $n-1$. A set of successor permutations $\mathcal{P}'$ of P is defined as $\mathcal{P}' = \{P'_i \mid i \in [1,n]\}$ where P'_i evolves from P by incrementing each element of P by one and inserting the missing 1 at position i, such that each position j in P corresponds to a position j' in P'_i:*

$$\begin{aligned} & j' = j, && \text{if } j < i. \\ \text{and} \quad & j' = j+1, && \text{if } j \geq i, \end{aligned}$$

and

$$\begin{aligned} & P'_i[j'] = P[j] + 1, && \text{if } j' \neq i \\ \text{and} \quad & P'_i[j'] = 1, && \text{if } j' = i. \end{aligned}$$

The insertion at position i shifts the elements at positions j, $j \geq i$, to the right resulting in an increased rank for the respective elements of P'_i.

Lemma 5.2. *Let P be a permutation of length $n-1$ and $P' = P'_i$ a successor of P with insertion position i, then we have for all $e \in [1, n-1]$ that*

(a) $R'[e+1] = R[e]$ *if* $R[e] < i$,

(b) $R'[e+1] = R[e]+1$ *if* $R[e] \geq i$, *and*

(c) $R'[1] = i$.

Proof. Let e be an arbitrary element of the permutation P occurring at position j, $e = P[j]$ and $R[e] = j$.

(a) If $R[e] < i$ then $j = R[e] < i$. Therefore, according to Definition (5.1), j' equals j and hence $P'[j'] = P[j] + 1 = e + 1$. Altogether, this gives $R'[e+1] = R'[P'[j']] = j' = j = R[e]$.

(b) If $R[e] \geq i$ then $j = R[e] \geq i$. Therefore, $j' = j+1$ and $P'[j'] = P[j]+1 = e+1$. This gives $R'[e+1] = R'[P'[j']] = j' = j+1 = R[e]+1$.

(c) $R'[1] = i$ holds because 1 is inserted at position i, $P'[i] = 1$.

In this way, the insertion position i determines the rank array R' of the successor permutation. □

Furthermore, mapping P to P' basically preserves the R_+-order:

Lemma 5.3. *Let P be a permutation of length $n-1$ with successor P'. For all indices g and h, $g, h \in [1, n-1]$,*

$$R_+[g] < R_+[h] \Longrightarrow R'_+[g'] < R'_+[h'].$$

Proof. Let g and h be some positions of P such that $R_+[g] < R_+[h]$. Then, according to the definition of R_+, $R[P[g]+1] < R[P[h]+1]$. We distinguish two cases.

(i) If $R[P[g]+1] < i$ then Lemma 5.2 (a and b) gives

$$R'[P[g]+1+1] = R[P[g]+1] < R[P[h]+1] \leq R'[P[h]+1+1].$$

Combining this with Definition 5.1 and the definition of R'_+ yields

$$R'_+[g'] = R'[P'[g']+1] < R'[P'[h']+1] = R'_+[h'].$$

(ii) If $R[P[g]+1] \geq i$ the proof works analogously using the fact that $R[P[h]+1] > R[P[g]+1] \geq i$. Hence, Lemma 5.2(b) has to be used for $R[P[g]+1]$ as well as for $R[P[h]+1]$, and then the rest of the proof proceeds as before.

Thus, except for the insertion position i, the R_+-order of P determines the R_+-order of P'. □

Lemma 5.3 considers the R_+-order of P', but leaves out the insertion position i. The next lemma states that the R_+-order at position i just depends on the position $R[1]$ of element 1 in the permutation P.

Lemma 5.4. *Let P' be a successor of P with insertion position i and g an index of P, then*

$$R_+[g] < R[1] \Longleftrightarrow R'_+[g'] < R'_+[i] \qquad \text{for all } g \in [1, n-1].$$

Proof. We first show that $R_+[g] < R[1] \Longrightarrow R'_+[g'] < R'_+[i]$.

If $R_+[g] < R[1]$ then using the definition of R_+ leads to $R[P[g]+1] < R[1]$. We consider two cases.

(i) If $R[P[g]+1] < i$ then applying Lemma 5.2 implicates $R'[P[g]+1+1] = R[P[g]+1]$ and $R[1] \leq R'[1+1]$. This together leads to

$$R'[(P[g]+1)+1] < R'[1+1]. \tag{8}$$

According to Definition 5.1, $P'[g'] = P[g]+1$ and $P'[i] = 1$. Combining this and inequality (8) leads to

$$R'[P'[g']+1] < R'[P'[i]+1].$$

Finally, according to the definition of R'_+, $R'_+[g'] < R'_+[i]$.

(ii) If $R[P[g]+1] \geq i$ then the proof proceeds analogously by considering $R[1] > R[P[g]+1] \geq i$.

Also, $R_+[g] > R[1] \Longrightarrow R'_+[g'] > R'_+[i]$. Since, for all $g \in [1, n-1]$, $R_+[g] \neq R[1]$ and $R'_+[g'] \neq R'_+[i]$, we finally obtain the stated equivalence. □

After characterizing the R_+-order of successor permutations, we now prove that through the mapping from P to an arbitrary successor permutation the number of R_+-descents is preserved or increased by one.

Lemma 5.5. *Let P be a permutation of length $n-1$ with d R_+-descents and $\mathcal{P}'$ the set of successor permutations for P, then for all successor permutations $P'_i \in \mathcal{P}'$, we have*

$$|\mathrm{desc}(P)| \leq |\mathrm{desc}(P'_i)| \leq |\mathrm{desc}(P)| + 1.$$

Proof. According to Lemma 5.3, the mapping with respect to insertion position i does not touch the R_+-order of consecutive positions not adjacent to i. More precisely, for all $j \in [2, n-1]$ with $j \neq i$

$$R_+[j-1] > R_+[j] \Longleftrightarrow R'_+[(j-1)'] > R'_+[j']. \tag{9}$$

That means, each R_+-descent at position $j-1$, $j \neq i$, corresponds to an R_+-descent at position $(j-1)'$ in P'_i and vice versa. Therefore, we just have to examine the R_+-order of the remaining pair of positions $(i-1, i)$ in P and the respective interval $[(i-1)', i']$ of P'_i. Note that $[(i-1)', i'] = \{i-1, i, i+1\}$. We distinguish the two cases that either position $i-1$ of P is an R_+-descent, or not.

(i) If $i-1$ is an R_+-descent of P so that $R_+[i-1] > R_+[i]$, then applying Lemma 5.3 gives

$$R'_+[(i-1)'] > R'_+[i']. \tag{10}$$

Since $R[1] \neq R_+[f]$ for all $f \in [1, n-1]$, we consider three subcases:

(i.1) If $R[1] > R_+[i-1]$ then Lemma 5.4 implies $R'_+[i] > R'_+[(i-1)']$ and together with inequality (10) $R'_+[i] > R'_+[(i-1)'] > R'_+[i']$ follows. That is, $R'_+[(i-1)'] < R'_+[i]$ and $R'_+[i] > R'_+[i']$. Hence, i is an R_+-descent of P'_i and the number of R_+-descents of P'_i equals the number of R_+-descents of P.

(i.2) If $R_+[i-1] > R[1] > R_+[i]$ then Lemma 5.4 implies $R'_+[(i-1)'] > R_+[i] > R_+[i']$. Hence, $(i-1)'$ and i are R_+-descents of P'_i. The number of R_+-descents in P'_i is thus one more than in P.

(i.3) If $R_+[i] > R[1]$ then $R'_+[(i-1)'] > R_+[i] < R_+[i']$. Hence, the number of R_+-descents in P'_i equals the number of R_+-descents in P.

(ii) If i is not an R_+-descent of P then three different cases analogously to (i) also yield that the number of R_+-descents retains or increases by one, respectively.

Combining all these cases tells, for each i, the mapping of P to P'_i preserves or increases the number of R_+-descents by one, respectively. □

Lemma 5.6. *Let P be a permutation with d R_+-descents and $\mathcal{P}'$ the set of successor permutations for P, then the number of successor permutations with d R_+-descents is $d+1$,*

$$d+1 = |\{P' \in \mathcal{P}' \mid |\mathrm{desc}(P')| = d\}|.$$

Proof. Let P be the permutation of length n and $\mathrm{desc}(P)$ the R_+-descent set of P of cardinality d, $d = |\mathrm{desc}(P)|$. The set of R_+-runs, $\mathrm{runs}(P)$, is implicitly defined by the set of R_+-descents, and also $|\mathrm{runs}(P)| = |\mathrm{desc}(P)|+1 = d+1$.

To each run $[g,h]$ of P, we assign a so called *proper insertion position* i, $i \in [g, h+1]$, preserving the number of R_+-descents through the mapping from P to P'_i, and show that the number of R_+-descents increases for the other, not-proper insertion positions.

Let now (g,h) be an R_+-run defined by a pair of consecutive R_+-descents, $(g-1,h)$, such that $R_+[g-1] > R_+[g] < R_+[g+1] < \ldots < R_+[h] > R_+[h+1]$.

Remember, according to Lemma 5.3, the R_+-descents not adjacent to the insertion position are preserved through the mapping to P'_i. Therefore, it suffices to investigate the R_+-order of positions touched by the insertion.

Since $R[1] \neq R_+[f]$ for all $f \in [1,n]$, we consider three mutually exclusive cases.

(i) If $R[1] < R_+[g]$ then the proper insertion position is g, $i = g$, such that

$$R_+[g-1] > R[1] < R_+[g] < \ldots < R_+[h] > R_+[h+1].$$

According to Lemmas 5.3 and 5.4, we get the series of inequalities

$$R_+[(g-1)'] > R'_+[i] < R'_+[g'] < \ldots < R_+[h'] > R_+[(h+1)'].$$

Hence, for the insertion position g, there exist exactly as many R_+-descents in the interval $[g,h]$ of P as in the interval $[g',h']$ and according to Lemma 5.3 the other R_+-descents are not affected through the mapping. Thus, $|\mathrm{desc}(P)| = |\mathrm{desc}(P'_i)|$.

For the insertion positions $i \in [g+1, h]$,

$$\begin{aligned} &R_+[g] > R_+[g+1] < \ldots \\ &\quad \ldots < R_+[i-1] > R[1] < R_+[i] < \ldots < R_+[h] > R_+[h+1] \end{aligned}$$

holds. Then, applying Lemmas 5.3 and 5.4 leads to

$$R_+[g'] > R'_+[(g+1)'] < \ldots$$
$$\ldots < R_+[(i-1)'] > R'_+[i] < R'_+[i'] < \ldots < R'_+[h'] > R_+[(h+1)'].$$

Therefore, the number of R_+-descents increases through the mapping.

We are left over with the bordering insertion position $h+1$, for which we consider two special cases.

(i.1) If $R[1] < R_+[h+1]$ then $h+1$ would be the proper insertion position for the next run $(h+1, l)$ for some l, like case (i).

(i.2) If $R[1] > R_+[h+1]$ then the insertion position $h+1$ increases the number of R_+-descents through the mapping from P to P'_i.

(ii) If $R_+[g] < R[1] < R_+[h]$ then analogously i, $i \in [g+1, h]$, with $R_+[i-1] < R[1] < R_+[i]$ is the proper insertion position. The other insertion positions j, $j \in [g+1, h]$ with $j \neq i$, are increasing the number of R_+-descents, and the bordering insertion positions g or $h+1$, respectively, either increase the number of R_+-descents analogously to (i.2), or they are proper insertion positions for the adjacent runs.

(iii) If $R_+[h] < R[1]$ then the proof works analogously to (i) by handling the bordering insertion position g like (i.2).

So far, we concentrated on the inner runs (g, h), with $g \neq 1$ and $h \neq n$. For the bordering runs (g, h) with $g = 1$ or $h = n$, respectively, the proper insertion positions are defined in the same way. Just the proof proceeds a bit simpler, because the insertion positions at the borders 1 and $n+1$, respectively, are not affected by adjacent runs.

Finally, for each of the $d+1$ runs in P, there exists a unique insertion position i that preserves the number of R_+-descents through the mapping from P to P'_i. All other insertion positions increase the number of R_+-descents. □

Theorem 5.7. *Let $A(n, d)$ be the number of permutations of length n with d R_+-descents, then*

$$A(n, d) = \left\langle {n \atop d} \right\rangle.$$

Proof.

(i) Since the permutation $(n, n-1, \ldots, 1)$ is the only one without any R_+-descent, $A(n, 0) = 1$.

(ii) Obviously, the number of potential R_+-descents is limited by $n-1$. Hence, there is no permutation of length n with more than $n-1$ R_+-descents, and thus $A(n, d) = 0$ for $d \geq n$.

(iii) As mentioned before, mapping each permutation P of length n to P'_i with all possible insertion positions i leads to n successor permutations each of length n. If P contains d R_+-descents, then Lemma 5.6 implies: there exist exactly $d+1$ successor permutations with d R_+-descents and, according to Lemma 5.5, the other $n-d$ successors permutations contain $d+1$ R_+-descents. Combining these observations leads to the recursion $A(n,d) = (d+1)A(n-1,d) + (n-d)A(n-1,d-1)$ for $0 < d < n$.

The propositions (i),(ii), and (iii) yield the same recursion as for the Eulerian numbers. Hence, $A(n,d) = \left\langle {n \atop d} \right\rangle$. □

Bannai *et al.* [2] showed that each suffix array with d R_+-descents can be associated with a string of at least $d+1$ different characters. Therefore, we sum up the appropriate suffix arrays to obtain the number of suffix arrays for strings over a fixed size alphabet.

Corollary 5.8. *Let Σ be a fixed size alphabet, $\sigma = |\Sigma|$. The number of distinct suffix arrays of length n for strings over Σ is $\sum_{d=0}^{\sigma-1} \left\langle {n \atop d} \right\rangle$.*

Proof. After Bannai *et al.* [2], all suffix arrays with up to $\sigma - 1$ R_+-descents have at least one string with no more than σ characters. □

Many application areas for suffix arrays handle small alphabets like the DNA, amino acid, or ASCII alphabet. Corollary 5.8 thus limits the number of distinct suffix arrays for such applications. For a DNA alphabet of size 4, for example, the number of distinct suffix arrays of length 15 is $861,948,404 = \sum_{d=0}^{3} \left\langle {15 \atop d} \right\rangle$, whereas the number of possible permutations of length 15 is $1,307,674,368,000 = 15!$ which is about $1,517$ times larger, and this difference rapidly increases for larger n.

Moreover, we achieve a lower bound on the compressibility of the whole information content of suffix arrays.

Corollary 5.9. *For strings of length n over an alphabet of size σ, the lower bound for the compressibility of their suffix arrays in the Kolmogorov sense is $\log \sum_{d=0}^{\sigma-1} \left\langle {n \atop d} \right\rangle$.*

Proof. There are $\sum_{d=0}^{\sigma-1} \left\langle {n \atop d} \right\rangle$ distinct suffix arrays. Among them, there exists at least one binary representation with Kolmogorov complexity not less than $\log \sum_{d=0}^{\sigma-1} \left\langle {n \atop d} \right\rangle$. □

6 Summation Identities.

We present constructive proofs for two long known summation identities of Eulerian numbers deduced by summing up the number of different suffix arrays for fixed alphabet size and string length. We believe that our constructive proofs are simpler than previous ones.

The identity $\sigma^n = \sum_i \left\langle {n \atop i} \right\rangle \binom{\sigma+i}{n}$, as given in [10, eq. 6.37], was proved by J. Worpitzki, already in 1883. We prove it by summing up the number of distinct strings of length n over a given alphabet of size σ for each suffix array:

$$\begin{aligned}
\sigma^n &= \sum_{d=0}^{\sigma-1} \left\langle {n \atop d} \right\rangle \binom{n+\sigma-d-1}{\sigma-d-1} && (11)\\
&= \sum_{d=0}^{\sigma-1} \left\langle {n \atop n-1-d} \right\rangle \binom{n+\sigma-d-1}{n} && (12)\\
&= \sum_{i=n-\sigma}^{n-1} \left\langle {n \atop i} \right\rangle \binom{\sigma+i}{n} && (13)\\
&= \sum_{i} \left\langle {n \atop i} \right\rangle \binom{\sigma+i}{n}. && (14)
\end{aligned}$$

Equality (12) follows from the symmetry rule for Eulerian and binomial numbers, equality (13) from substituting $i = n-d-1$, and equality (14) from $\left\langle {n \atop i} \right\rangle = 0$ for all $i \geq n$ and $\binom{a+i}{n} = 0$ for all $i < n-\sigma$, respectively.

The second summation identity, which we are concerned with, is the summation rule for Eulerian numbers to generate the Stirling numbers of the second kind [10, Eq. 6.39]: $k!\left\{ {n \atop k} \right\} = \sum_i \left\langle {n \atop i} \right\rangle \binom{i}{n-k}$. To prove this identity, we count the $k!\left\{ {n \atop k} \right\}$ strings composed of exactly k different characters. Summing up these strings for each suffix array gives

$$\begin{aligned}
k!\left\{ {n \atop k} \right\} &= \sum_{d=0}^{k-1} \left\langle {n \atop d} \right\rangle \binom{n-d-1}{k-d-1} && (15)\\
&= \sum_{d} \left\langle {n \atop d} \right\rangle \binom{(n-k)+(k-d-1)}{k-d-1} && (16)\\
&= \sum_{d} \left\langle {n \atop n-1-d} \right\rangle \binom{n-d-1}{n-k} && (17)\\
&= \sum_{i} \left\langle {n \atop i} \right\rangle \binom{i}{n-k}. && (18)
\end{aligned}$$

Equality (16) holds since $\left\langle {n \atop d} \right\rangle = 0$ for $d \geq k$, equality (17) follows from the symmetry rule for Eulerian and binomial numbers, and equality (18) from substituting $d = n-1-i$.

7 Conclusion

We have presented constructive proofs to count the strings sharing the same suffix array as well as the distinct suffix arrays for fixed size alphabets. For

alphabets of size σ, $\binom{n+\sigma-d-1}{\sigma-d-1}$ strings share the same suffix array (with d R_+-descents) among which $\binom{n-d-1}{\sigma-d-1}$ are composed of exactly σ distinct characters. For these strings we have given a bijection into the set of non-decreasing sequences over $\sigma - d$ integers. The number of distinct suffix arrays is $\sum_{d=0}^{\sigma-1} \left\langle {n \atop d} \right\rangle$. This has yielded a $\log \sum_{d=0}^{\sigma-1} \left\langle {n \atop d} \right\rangle$ lower bound for the compressibility of such suffix arrays.

Moreover, summing up the number of strings for each suffix array yields constructive proofs for Worpitzki's identity and for the summation rule of Eulerian numbers to generate the Stirling numbers of the second kind, respectively. One could also say the number of suffix arrays and its strings form a particular instance of these identities.

Of further interest will be the development of efficient enumeration algorithms for which our constructive proofs have already suggested suitable methods. For the enumeration of strings sharing the same suffix array, we have proved the equivalence to the enumeration of non-decreasing sequences which can be easily performed in optimal time, whereas the enumeration of distinct suffix arrays in optimal time requires further development.

Acknowledgments. We thank Veli Mäkinen, Hans-Michael Kaltenbach, and Constantin Bannert for helpful discussions.

References

[1] Mohamed I. Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.

[2] Hideo Bannai, Shunsuke Inenaga, Ayumi Shinohara, and Masayuki Takeda. Inferring strings from graphs and arrays. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science (MFCS 2003)*, volume 2747 of *LNCS*, pages 208–217. Springer Verlag, 2003.

[3] Stefan Burkhardt and Juha Kärkkäinen. Fast lightweight suffix array construction and checking. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, volume 2676 of *LNCS*, pages 55–69. Springer Verlag, 2003.

[4] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital System Research Center, 1994.

[5] Maxime Crochemore, Jacques Désarménien, and Dominique Perrin. A note on the Burrows-Wheeler transformation. *Theoretical Computer Science*, 332(1-3):567–572, 2005.

[6] Erik D. Demaine and Alejandro López-Ortiz. A linear lower bound on index size for text retrieval. *Journal of Algorithms*, 48(1):2–15, 2003.

[7] Jean-Pierre Duval and Arnaud Lefebvre. Words over an ordered alphabet and suffix permutations. *RAIRO – Theoretical Informatics and Applications*, 36(3):249–259, 2002.

[8] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000)*, pages 390–398. IEEE Computer Society, 2000.

[9] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. New indices for text: Pat trees and pat arrays. In W. B. Frakes and Ricardo A. Baeza-Yates, editors, *Information retrieval: data structures and algorithms*, pages 66–82. Prentice-Hall, 1992.

[10] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, second edition, 1994.

[11] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2000)*, pages 397–406, 2000.

[12] Meng He, J. Ian Munro, and S. Srinivasa Rao. A categorization theorem on suffix arrays with applications to space efficient text indexes. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*, pages 23–32. SIAM, 2005.

[13] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[14] Peter Bro Miltersen. Lower bounds on the size of selection and rank indexes. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*, pages 11–12. SIAM, 2005.

[15] Dennis Moore, William F. Smyth, and Dianne Miller. Counting distinct strings. *Algorithmica*, 23(1):1–13, 1999.

[16] Kunihiko Sadakane. Compressed text databases with efficient query algorithms based on the compressed suffix array. In *Proceedings of the 11th International Symposium on Algorithms and Computation (ISAAC 2000)*, volume 1969 of *LNCS*, pages 410–421. Springer Verlag, 2000.

Bisher erschienene Reports an der Technischen Fakultät
Stand: 2005-07-25

94-01 Modular Properties of Composable Term Rewriting Systems
(Enno Ohlebusch)

94-02 Analysis and Applications of the Direct Cascade Architecture
(Enno Littmann, Helge Ritter)

94-03 From Ukkonen to McCreight and Weiner: A Unifying View of Linear-Time Suffix Tree Construction
(Robert Giegerich, Stefan Kurtz)

94-04 Die Verwendung unscharfer Maße zur Korrespondenzanalyse in Stereo Farbbildern
(André Wolfram, Alois Knoll)

94-05 Searching Correspondences in Colour Stereo Images – Recent Results Using the Fuzzy Integral
(André Wolfram, Alois Knoll)

94-06 A Basic Semantics for Computer Arithmetic
(Markus Freericks, A. Fauth, Alois Knoll)

94-07 Reverse Restructuring: Another Method of Solving Algebraic Equations
(Bernd Bütow, Stephan Thesing)

95-01 PaNaMa User Manual V1.3
(Bernd Bütow, Stephan Thesing)

95-02 Computer Based Training-Software: ein interaktiver Sequenzierkurs
(Frank Meier, Garrit Skrock, Robert Giegerich)

95-03 Fundamental Algorithms for a Declarative Pattern Matching System
(Stefan Kurtz)

95-04 On the Equivalence of E-Pattern Languages
(Enno Ohlebusch, Esko Ukkonen)

96-01 Static and Dynamic Filtering Methods for Approximate String Matching
(Robert Giegerich, Frank Hischke, Stefan Kurtz, Enno Ohlebusch)

96-02 Instructing Cooperating Assembly Robots through Situated Dialogues in Natural Language
(Alois Knoll, Bernd Hildebrand, Jianwei Zhang)

96-03 Correctness in System Engineering
(Peter Ladkin)

96-04 An Algebraic Approach to General Boolean Constraint Problems
(Hans-Werner Güsgen, Peter Ladkin)

96-05 Future University Computing Resources
(Peter Ladkin)

96-06 Lazy Cache Implements Complete Cache
(Peter Ladkin)

96-07 Formal but Lively Buffers in TLA+
(Peter Ladkin)

96-08 The X-31 and A320 Warsaw Crashes: Whodunnit?
(Peter Ladkin)

96-09 Reasons and Causes
(Peter Ladkin)

96-10 Comments on Confusing Conversation at Cali
(Dafydd Gibbon, Peter Ladkin)

96-11 On Needing Models
(Peter Ladkin)

96-12 Formalism Helps in Describing Accidents
(Peter Ladkin)

96-13 Explaining Failure with Tense Logic
(Peter Ladkin)

96-14 Some Dubious Theses in the Tense Logic of Accidents
(Peter Ladkin)

96-15 A Note on a Note on a Lemma of Ladkin
(Peter Ladkin)

96-16 News and Comment on the AeroPeru B757 Accident
(Peter Ladkin)

97-01 Analysing the Cali Accident With a WB-Graph
(Peter Ladkin)

97-02 Divide-and-Conquer Multiple Sequence Alignment
(Jens Stoye)

97-03 A System for the Content-Based Retrieval of Textual and Non-Textual Documents Based on Natural Language Queries
(Alois Knoll, Ingo Glöckner, Hermann Helbig, Sven Hartrumpf)

97-04 Rose: Generating Sequence Families
(Jens Stoye, Dirk Evers, Folker Meyer)

97-05 Fuzzy Quantifiers for Processing Natural Language Queries in Content-Based Multimedia Retrieval Systems
(Ingo Glöckner, Alois Knoll)

97-06 DFS – An Axiomatic Approach to Fuzzy Quantification
(Ingo Glöckner)

98-01 Kognitive Aspekte bei der Realisierung eines robusten Robotersystems für Konstruktionsaufgaben
(Alois Knoll, Bernd Hildebrandt)

98-02 A Declarative Approach to the Development of Dynamic Programming Algorithms, applied to RNA Folding
(Robert Giegerich)

98-03 Reducing the Space Requirement of Suffix Trees
(Stefan Kurtz)

99-01 Entscheidungskalküle
(Axel Saalbach, Christian Lange, Sascha Wendt, Mathias Katzer, Guillaume Dubois, Michael Höhl, Oliver Kuhn, Sven Wachsmuth, Gerhard Sagerer)

99-02 Transforming Conditional Rewrite Systems with Extra Variables into Unconditional Systems
(Enno Ohlebusch)

99-03 A Framework for Evaluating Approaches to Fuzzy Quantification
(Ingo Glöckner)

99-04 Towards Evaluation of Docking Hypotheses using elastic Matching
(Steffen Neumann, Stefan Posch, Gerhard Sagerer)

99-05 A Systematic Approach to Dynamic Programming in Bioinformatics. Part 1 and 2: Sequence Comparison and RNA Folding
(Robert Giegerich)

99-06 Autonomie für situierte Robotersysteme – Stand und Entwicklungslinien
(Alois Knoll)

2000-01 Advances in DFS Theory
(Ingo Glöckner)

2000-02 A Broad Class of DFS Models
(Ingo Glöckner)

2000-03 An Axiomatic Theory of Fuzzy Quantifiers in Natural Languages
(Ingo Glöckner)

2000-04 Affix Trees
(Jens Stoye)

2000-05 Computergestützte Auswertung von Spektren organischer Verbindungen
(Annika Büscher, Michaela Hohenner, Sascha Wendt, Markus Wiesecke, Frank Zöllner, Arne Wegener, Frank Bettenworth, Thorsten Twellmann, Jan Kleinlützum, Mathias Katzer, Sven Wachsmuth, Gerhard Sagerer)

2000-06 The Syntax and Semantics of a Language for Describing Complex Patterns in Biological Sequences
(Dirk Strothmann, Stefan Kurtz, Stefan Gräf, Gerhard Steger)

2000-07 Systematic Dynamic Programming in Bioinformatics (ISMB 2000 Tutorial Notes)
(Dirk J. Evers, Robert Giegerich)

2000-08 Difficulties when Aligning Structure Based RNAs with the Standard Edit Distance Method
(Christian Büschking)

2001-01 Standard Models of Fuzzy Quantification
(Ingo Glöckner)

2001-02 Causal System Analysis
(Peter B. Ladkin)

2001-03 A Rotamer Library for Protein-Protein Docking Using Energy Calculations and Statistics
(Kerstin Koch, Frank Zöllner, Gerhard Sagerer)

2001-04 Eine asynchrone Implementierung eines Microprozessors auf einem FPGA
(Marco Balke, Thomas Dettbarn, Robert Homann, Sebastian Jaenicke, Tim Köhler, Henning Mersch, Holger Weiss)

2001-05 Hierarchical Termination Revisited
(Enno Ohlebusch)

2002-01 Persistent Objects with O2DBI
(Jörn Clausen)

2002-02 Simulation von Phasenübergängen in Proteinmonoschichten
(Johanna Alichniewicz, Gabriele Holzschneider, Morris Michael, Ulf Schiller, Jan Stallkamp)

2002-03 Lecture Notes on Algebraic Dynamic Programming 2002
(Robert Giegerich)

2002-04 Side chain flexibility for 1:n protein-protein docking
(Kerstin Koch, Steffen Neumann, Frank Zöllner, Gerhard Sagerer)

2002-05 ElMaR: A Protein Docking System using Flexibility Information
(Frank Zöllner, Steffen Neumann, Kerstin Koch, Franz Kummert, Gerhard Sagerer)

2002-06 Calculating Residue Flexibility Information from Statistics and Energy based Prediction
(Frank Zöllner, Steffen Neumann, Kerstin Koch, Franz Kummert, Gerhard Sagerer)

2002-07 Fundamentals of Fuzzy Quantification: Plausible Models, Constructive Principles, and Efficient Implementation
(Ingo Glöckner)

2002-08 Branching of Fuzzy Quantifiers and Multiple Variable Binding: An Extension of DFS Theory
(Ingo Glöckner)

2003-01 On the Similarity of Sets of Permutations and its Applications to Genome Comparison
(Anne Bergeron, Jens Stoye)

2003-02 SNP and mutation discovery using base-specific cleavage and MALDI-TOF mass spectrometry
(Sebastian Böcker)

2003-03 From RNA Folding to Thermodynamic Matching, including Pseudoknots
(Robert Giegerich, Jens Reeder)

2003-04 Sequencing from compomers: Using mass spectrometry for DNA de-novo sequencing of 200+ nt
(Sebastian Böcker)

2003-05 Systematic Investigation of Jumping Alignments
(Constantin Bannert)

2003-06 Suffix Tree Construction and Storage with Limited Main Memory
(Klaus-Bernd Schürmann, Jens Stoye)

2003-07 Sequencing from compomers in thepresence of false negative peaks
(Sebastian Böcker)

2003-08 Genalyzer: An Interactive Visualisation Tool for Large-Scale Sequence Matching – Biological Applications and User Manual
(Jomuna V. Choudhuri, Chris Schleiermacher)

2004-01 Sequencing From Compomers is NP-hard
(Sebastian Böcker)

2004-02 The Money Changing Problem revisited: Computing the Frobenius number in time $O(k\, a_1)$
(Sebastian Böcker, Zsuzsanna Lipták)

2004-03 Accelerating the Evaluation of Profile HMMs by Pruning Techniques
(Thomas Plötz, Gernot A. Fink)

2004-04 Optimal Group Testing Strategies with Interval Queries and Their Application to Splice Site Detection
(Ferdinando Cicalese, Peter Damaschke, Ugo Vaccaro)

2004-05 Compressed Representation of Sequences and Full-Text Indexes
(Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, Gonzalo Navarro)

2005-01 Overlaps Help: Improved Bounds for Group Testing with Interval Queries
(Ferdinando Cicalese, Peter Damaschke, Libertad Tansini, Sören Werth)

2005-02 Two batch Fault-tolerant search with error cost constraints: An application to learning
(Ferdinando Cicalese)

2005-03 Searching for the Shortest Common Supersequence
(Sergio A. de Carvalho Jr., Sven Rahmann)