A computational substantiation of the d-step approach to the number of distinct squares problem

Antoine Deza, Frantisek Franek, Mei Jiang

Advanced Optimization Laboratory
Department of Computing and Software
McMaster University, Hamilton, Ontario, Canada

Abstract

Motivated by the recent validation of the d-step approach for the number of runs problem, we investigate the largest possible number $\sigma_d(n)$ of distinct primitively rooted over all strings of length n with exactly d distinct symbols. New properties of $\sigma_d(n)$ are presented, and the notion of s-cover is introduced with an emphasis on the recursive computational determination of $\sigma_d(n)$. In particular, we were able to determine all values of $\sigma_2(n)$ for $n \leq 70$, $\sigma_3(n)$ for $n \leq 45$ and $\sigma_4(n)$ for $n \leq 38$. These computations reveal the unexpected existence of pairs (d, n) satisfying $\sigma_{d+1}(n+2) - \sigma_d(n) > 1$ such as (2,33) and (2,34), and of three consecutive equal values: $\sigma_2(31) = \sigma_2(32) = \sigma_2(33)$. Noticeably, we show that among all strings of length 33, the maximum number of distinct primitively rooted squares cannot be achieved by a non-ternary string.

Keywords: string, square, primitively rooted square, maximum number of distinct primitively rooted squares, parameterized approach, $(d, n-d)$ table

1. Introduction

The notion of an r-cover was introduced by Baker, Deza, and Franek [1] as a means to represent the distribution of the runs in a string and thus describe the structure of the run-maximal strings. Ignoring the number of distinct symbols d in the string, a key assertion states that essentially any

Email addresses: deza@mcmaster.ca (Antoine Deza), franek@mcmaster.ca (Frantisek Franek), jiangm5@mcmaster.ca (Mei Jiang)

run-maximal string has an r-cover. This fact was used in [2] to compute values of the maximum number of runs for strings of previously intractable lengths, and to provide computational substantiation for the d-step approach to the problem of the maximum number of runs proposed by Deza and Franek [4]. Recently, Bannai et al. [3] proved that the number of runs in a string is at most its length minus 3 using the maximal Lyndon roots of runs. Considering the largest possible number $\rho_d(n)$ of runs over all strings of length n with exactly d distinct symbols, Deza and Franek [4] conjectured that $\rho_d(n) \leq n - d$ and $\rho_d(n) \leq n - d - 1$ for $n \geq 2d + 1$ which was proven by Bannai et al. [3]. The bound was slightly improved to $\rho_d(n) \leq n - d - 2$ for $n \geq 2d+5$ by Deza and Franek [5] and, consequently, the number of runs in a string of length at least 9 is at most its length minus 4. Fischer, Holub, I, and Lewenstein further exploited the maximal Lyndon root approach and strengthened the upper for the maximum number of runs for binary strings in [9].

In this paper, we present a method of computing square-maximal strings similar to the one used for runs in [2] and similarly based on the d-step approach. We introduce the notion of s-cover which is used to speed up computations of the maximum number of distinct primitively rooted squares allowing computing $\sigma_d(n)$ for previously intractable values of d and n. The paper is organized as follows: Section 2 gives the basic facts and notation, Section 3 discusses the computational approach to the number of distinct primitively rooted squares, Section 4 introduces a heuristic for speeding up the computation, Section 5 discusses how s-covered string can be generated, Section 6 discusses how to compute $\sigma_d(n)$ values, Section 7 discusses how to compute $\sigma_d(2d)$ values. In Section 8 some additional theoretic properties of $\sigma_d(n)$ not presented in [7] are discussed. The computational results are summarized in Section 9.

2. Notations

We encode a square as a triple (s, e, p) where s is the starting position of the square, e is the ending position of the square, and p is its period. Note that $e = s + 2p - 1$. The *join* $x[i_1..\, i_k] \ \vee \ x[j_1..\, j_m]$ of two substrings of a string $x = x[1..\, n]$ is defined if $i_1 \leq j_1 \leq i_k + 1$ and then $x[i_1..\, i_k] \ \vee \ x[j_1..\, j_m] = x[i_1..\, max\{i_k, j_m\}]$, or if $j_1 \leq i_1 \leq j_m + 1$ and then $x[i_1..\, i_k] \ \vee \ x[j_1..\, j_m] = x[j_1..\, max\{i_k, j_m\}]$. In other words, the join is defined when the two substrings either are adjacent or overlapping. The join $S_1 \vee S_2$ of two

squares of x encoded as $S_1 = (s_1, e_1, p_1)$ and $S_2 = (s_2, e_2, p_2)$ is defined as the join $x[s_1..e_1] \vee x[s_2..e_2]$. The alphabet of x is denoted by $\mathcal{A}(x)$, (d, n)-string refers to a string of length n with exactly d distinct symbols, $\boldsymbol{s}(x)$ denotes the number of distinct primitively rooted squares in a string x, and $\sigma_d(n)$ refers to the maximum number of distinct primitively rooted squares over all (d, n)-strings. A singleton is a symbol which occurs exactly once in the string under consideration. For the empty string ε, we set $\boldsymbol{s}(\varepsilon) = 0$ and $\sigma_d(0) = 0$. In the d-step approach the main tool is the $(d, n-d)$ table of the $\sigma_d(n)$ values where the row index represents d while to column index represents $n-d$ rather than the usual n. A 20 x 20 fragment of the table with computed values is shown in Table 1, see [6] for a table with all currently computed values. An important aspect of the d-step approach is the fact that the bounding of $\sigma_d(n)$ is determined by the bounding on the main diagonal, i.e. $\sigma_d(2d)$. More precisely, $\sigma_d(n) \leq n - d$ for any $n \geq d \geq 2$ if and only if $\sigma_d(2d) = d$ for any $d \geq 2$. Additional properties $\sigma_d(n)$ are discussed and used in the following sections, see [4, 7] for details.

		$n-d$																		
		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
d	2	**2**	2	3	3	4	5	6	7	7	8	9	10	11	12	12	13	13	14	15
	3	2	**3**	3	4	4	5	6	7	8	8	9	10	11	12	13	13	14	14	15
	4	2	3	**4**	4	5	5	6	7	8	9	9	10	11	12	13	14	14	15	15
	5	2	3	4	**5**	5	6	6	7	8	9	10	10	11	12	13	14	15	15	16
	6	2	3	4	5	**6**	6	7	7	8	9	10	11	11	12	13	14	15	16	16
	7	2	3	4	5	6	**7**	7	8	8	9	10	11	12	12	13	14	15	16	17
	8	2	3	4	5	6	7	**8**	8	9	9	10	11	12	13	13	14	15	16	17
	9	2	3	4	5	6	7	8	**9**	9	10	10	11	12	13	14	14	15	16	17
	10	2	3	4	5	6	7	8	9	**10**	10	11	11	12	13	14	15	15	16	17
	11	2	3	4	5	6	7	8	9	10	**11**	11	12	12	13	14	15	16	16	17
	12	2	3	4	5	6	7	8	9	10	11	**12**	12	13	13	14	15	16	17	?
	13	2	3	4	5	6	7	8	9	10	11	12	**13**	13	14	14	15	16	17	18
	14	2	3	4	5	6	7	8	9	10	11	12	13	**14**	14	15	15	16	17	18
	15	2	3	4	5	6	7	8	9	10	11	12	13	14	**15**	15	16	16	17	18
	16	2	3	4	5	6	7	8	9	10	11	12	13	14	15	**16**	16	17	17	18
	17	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	**17**	17	18	18
	18	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**18**	18	19
	19	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	**19**	19
	20	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	**20**

Table 1: $(d, n-d)$ table for $\sigma_d(n)$ with $2 \leq d \leq 20$ and $2 \leq n - d \leq 20$ where the main diagonal corresponding to $n = 2d$ is shown in bold

3. Computational approach to distinct primitively rooted squares

In the computational framework for determining $\sigma_d(n)$ we will be discussing later, we first compute a lower bound of $\sigma_d(n)$ denoted as $\sigma_d^-(n)$. It is enough

to consider (d, n)-strings x that could achieve $\boldsymbol{s}(x) > \sigma_d^-(n)$ for determining $\sigma_d(n)$, thus significantly reducing the search space. The purpose of this section is to introduce the necessary conditions that guarantee that for such an x, $\boldsymbol{s}(x) > \sigma_d^-(n)$ for a given $\sigma_d^-(n)$. The necessary conditions are the existence of an *s-cover* and a sufficient *density* of the string, see Lemmas 5, 9, 10. The s-cover is guaranteed through generation, while the density is verified incrementally during the generation at the earliest possible stages. Note that the notion of s-cover, though similar to r-cover for runs [1, 2], is slightly different.

Definition 1. *An* s-cover *of a string* $x = x[1..n]$ *is a sequence of primitively rooted squares* $\{S_i = (s_i, e_i, p_i) \mid 1 \leq i \leq m\}$ *so that*

(1) *for any* $1 \leq i < m$, $s_i < s_{i+1} \leq e_i + 1$ *and* $e_i < e_{i+1}$, *i.e. two consecutive squares are either adjacent or overlapping;*

(2) $\bigvee_{1 \leq i \leq m} S_i = x$;

(3) *for any occurrence of square* S *in* x, *there is* $1 \leq i \leq m$ *so that* S *is a substring of* S_i, *denoted by* $S \subseteq S_i$.

Lemma 2. *The s-cover of a string is unique.*

Proof. Let us assume that we have two different s-covers of x, $\{S_i \mid 1 \leq i \leq m\}$ and $\{S'_j \mid 1 \leq j \leq k\}$. We shall prove by induction that they are identical. By Definition 1 (3), $S_1 \subseteq S'_1$ and, by the same argument, $S'_1 \subseteq S_1$, and thus $S_1 = S'_1$. Let the induction hypothesis be $S_i = S'_i$ for $1 \leq i \leq t$. If $\bigvee_{1 \leq i \leq t} S_i = x$, we have $t = m = k$ and we are done. Otherwise consider S_{t+1}. By Definition 1 (3), there is S'_v so that $S_{t+1} \subseteq S'_v$ and $v > t$. We need to show that $v = t + 1$. If not, then S_{t+1} would neither be a substring of S'_t nor of S'_{t+1} contradicting Definition 1 (3). Therefore $S_{t+1} \subseteq S'_{t+1}$. By the same argument, $S'_{t+1} \subseteq S_{t+1}$ and so $S_{t+1} = S'_{t+1}$. □

Lemma 3. *If a string admits an s-cover, then it is singleton free.*

Proof. Let $\{S_j \mid 1 \leq j \leq m\}$ be the s-cover of $x = x[1..n]$. For any $1 \leq i \leq n$, $x[i] \in S_t$ for some t by Definition 1 (2). Since S_t is a square, the symbol $x[i]$ occurs in x at least twice. □

Before we define what a *dense string* is, we first define the notion of a *core* of a square, similarly to the core of a run [2, 11]. For the problem of distinct squares, a core of a square is the set of indices formed by the intersection of the indices of all its occurrences in the string.

Definition 4. *The* core vector $k(x)$ *of a* (d,n)*-string* x *is defined by* $k_i(x) =$ *the number of cores of squares of* x *containing* i *for* $i = 1,\ldots,n$*. A singleton-free* (d,n)*-string* x *is* dense *if its core vector* $k(x)$ *satisfies* $k_i(x) > \sigma_d^-(n) - \boldsymbol{s}(x[1..\, i-1]) - m_i$ *for* $i = 1,\ldots,n$ *where* $m_i = \max\{\sigma_{d'}(n-i) : d - |\mathcal{A}(x[1..\, i-1])| \le d' \le \min(n-i,d)\}$.

Lemma 5. *If a* (d,n)*-string* x *is not dense, then* $\boldsymbol{s}(x) \le \sigma_d^-(n)$.

Proof. The proof follows from the basic observation that for any string x, $\boldsymbol{s}(x) \le \boldsymbol{s}(x[1..\, i-1]) + \boldsymbol{s}(x[i+1..\, n]) + k_i(x)$ for any i. Note that the inequality occurs when there are the same type of squares in both $x[1..\, i-1]$ and $x[i+1..\, n]$. If x is not dense, then for some i_0, $k_{i_0}(x) \le \sigma_d^-(n) - \boldsymbol{s}(x[1..\, i_0-1]) - m_{i_0}$. Then $\boldsymbol{s}(x) \le \boldsymbol{s}(x[1..\, i_0-1]) + \boldsymbol{s}(x[i_0+1..\, n]) + k_{i_0}(x) \le \boldsymbol{s}(x[1..\, i_0-1]) + m_{i_0} + k_{i_0}(x) \le \boldsymbol{s}(x[1..\, i_0-1]) + m_{i_0} + \sigma_d^-(n) - \boldsymbol{s}(x[1..\, i_0-1]) - m_{i_0} = \sigma_d^-(n)$. □

Lemma 6. *If the core vector* $k(x)$ *of a* (d,n)*-string* x *satisfies* $k_i(x) > 0$ *for* $i = 1,\ldots,n$*, then* x *has an* s*-cover.*

Proof. We build an s-cover by induction: Since the $k_1(x) \ge 1$, 1 is in at least one core, hence there must be at least one square starting at position 1. Among all squares starting at position 1, set the one with the largest period to be S_1. Suppose that we have built the s-cover $\{S_i = (s_i, e_i, p_i) : i \le t\}$. If $\bigvee_{1\le i\le t} S_i = x$, we are done. Otherwise $\bigvee_{1\le i\le t} S_i = x[1..\, e_t]$ where $e_t < n$. Since $k_{e_t+1}(x) \ge 1$, there is at least one square (s,e,p) in x so that $s \le e_t + 1 \le s + 2p - 1$. From all such squares choose the leftmost ones, and among them choose the one with the largest period and set it as S_{t+1}. It is straightforward to verify that all the conditions of Definition 1 are satisfied and that we have built the s-cover of x. □

Note that for a (d,n)-string, having an s-cover implies being singleton free. However, it does not imply that every $k_i(x) \ge 1$, even though it is close to it. Consider the s-cover $\{S_j = (s_j, e_j, p_j) : 1 \le j \le m\}$ of x. If S_1 has another occurrence in x and there is no other square in x starting at position 1, then 1

is not in any core and $k_1(x) = 0$. Similarly, if the s-cover has two consecutive adjacent squares S_j and S_{j+1}, if there is no other square occurring at position s_{j+1}, and if the square S_{j+1} has some other occurrence, then $k_{s_{j+1}}(x) = 0$. In this sense, the s-cover is a computationally efficient structural generalization of the property that every $k_i(x) \geq 1$.

Lemma 7. *Let $\{S_i = (s_i, e_i, p_i) \mid 1 \leq i \leq m\}$ be an s-cover of x. Let $k = k(x)$ be the core vector of x. Then for $1 \leq i < m$, $1 \leq j < s_{i+1}$, $k_j(x[1..\, e_1]) \geq k_j(x)$.*

Proof. Let us assume that for some $i = 1, 2, \ldots, m-1$ there is a j so that $k_j(x) > k_j(x[1..\, e_i])$. Then there must exists a square (s, e, p) in $x = x[1..\, e_m]$ that is not a square of $x[1..\, e_i]$, i.e. $e > e_i$ and $s < s_{i+1}$, so it is an intermediate square violating the definition of s-cover, see Definition 1 (3). □

Lemma 8. *If a square-maximal (d, n)-string x has an s-cover with two consecutive adjacent squares, then $\sigma_d(n) \leq \sigma_{d_1}(n_1) + \sigma_{d_2}(n_2)$ for some $2 \leq d_1, d_2 \leq d \leq d_1 + d_2$ and some n_1, n_2, possibly equal to zero, such that $n_1 + n_2 = n$.*

Proof. Let $\{S_i : 1 \leq i \leq m\}$ be the s-cover of x and let $S_j \cap S_{j+1} = \emptyset$. Then $\boldsymbol{s}(x) \leq \boldsymbol{s}(x_1) + \boldsymbol{s}(x_2)$, where $x_1 = \bigvee_{1 \leq i \leq j} S_i$ and $x_2 = \bigvee_{j < i \leq m} S_i$. Therefore $\sigma_d(n) = \boldsymbol{s}(x) \leq \boldsymbol{s}(x_1) + \boldsymbol{s}(x_2) \leq \sigma_{d_1}(n_1) + \sigma_{d_2}(n_2)$ where x_1 and x_2 are, respectively, a (d_1, n_1)- and a (d_2, n_2)-string. □

Lemma 9. *If a singleton-free square-maximal (d, n)-string x does not have an s-cover, then $\sigma_d(n) = \sigma_d(n-1)$.*

Proof. Since x does not have an s-cover, there exist some i_0 such that $k_{i_0}(x) = 0$ by Lemma 6. Remove $x[i_0]$ to form a $(d, n-1)$-string y. This will not decrease the number of distinct squares in x since there is no core of any square containing i_0. Then $\sigma_d(n) = \boldsymbol{s}(x) \leq \boldsymbol{s}(y) \leq \sigma_d(n-1)$; that is, $\sigma_d(n) = \sigma_d(n-1)$ since $\sigma_d(n) \geq \sigma_d(n-1)$. □

Lemma 10. *If a square-maximal (d, n)-string has a singleton, then $\sigma_d(n) = \sigma_{d-1}(n-1)$.*

Proof. Remove the singleton to form a $(d-1, n-1)$-string y with $\sigma_d(n) = \boldsymbol{s}(x) \leq \boldsymbol{s}(y) \leq \sigma_{d-1}(n-1)$. Since $\sigma_d(n) \geq \sigma_{d-1}(n-1)$, see [7], therefore $\sigma_d(n) = \sigma_{d-1}(n-1)$. □

4. Heuristics for a lower bound $\boldsymbol{\sigma_d^-(n)}$

Recall that $\sigma_d^-(n)$ denotes the best available lower bound for $\sigma_d(n)$. The higher the value of $\sigma_d^-(n)$, the less computational effort must be spent on determining $\sigma_d(n)$. For $d = 2$, we generate $\mathcal{L}_2(n)$, the set of $(2, n)$-strings admitting an s-cover, being balanced over every prefix, having a maximum period bounded by at most a predefined constant, and containing no triples such that aaa or bbb. Note that the frequencies of a's and b's differ by at most a predefined constant. Thus, we set

$$\sigma_2^-(n) = \max\{\sigma_2(n-1), \max_{x \in \mathcal{L}_2(n)} \boldsymbol{s}(x)\}.$$

For $d \geq 3$, we set $\sigma_d^-(n) = \max\{\sigma_{d-1}(n-1), \sigma_{d-1}(n-2)+1, \sigma_d(n-1)\}$. The heuristic was found to be efficient as in almost all cases it gave the appropriate maximum value.

5. Generating special (d, n)-strings admitting an s-cover

Rather than generating strings, we generate their s-covers. By *special* in the title of this section we mean only s-covers that have no consecutive adjacent squares. The generation proceeds by extending the partially built s-cover in all possible ways. Every time a potential square of the s-cover is to be extended by one position, all previously used symbols and the first unused symbol are tried. For each symbol, the frequency counter is checked that the symbol does not exceed $n + 2 - 2d$. Once a symbol is used, the frequency counter is updated. When the whole s-cover is generated, the counter is checked whether all d symbols occurred in the resulting string; if not, the string is rejected. A typical implementation of the generation of the s-cover would be through recursion as backtracking is needed. For computational efficiency reasons we opted instead for a user-stack controlled backtracking implemented as a co-routine `Next()` allowing us to call the co-routine repeatedly to produce the next string. Note that the strings are generated in a lexicographic order. The generation of the s-cover follows

these principles: The generator for the first square is created by iterative calls to `Next()` producing all the possible generators. Each generator is checked for the additional properties – including must be primitive, did not create an intermediate square in the partial string – before it is accepted. For each subsequent square, its generator may be partially or fully determined. If it is partially determined, iterative calls to `Next()` are used to generate all possible completions of the generator. The complete generator is checked and accepted or rejected. In addition, if the density of the string being generated is to be checked, we use Lemma 7 and the core vector of the partially generated string to reject the string or allow it to be extended further.

6. Recursive computation of $\sigma_d(n)$

First, $\sigma_d^-(n)$ is computed by the heuristic of Section 4. Then it is verified that $\sigma_{d_1}(n_1)+\sigma_{d_2}(n_2) \le \sigma_d^-(n)$ for $2 \le d_1, d_2 \le d \le d_1+d_2$ and $n_1+n_2=n$. Then $\mathcal{U}_d(n)$, the set of all dense special (d,n)-strings admitting an s-cover is generated as described in Section 5. It follows that

$$\sigma_d(n) = \max\{\sigma_d^-(n), \max_{x\in\mathcal{U}_d(n)} \boldsymbol{s}(x)\}.$$

To see that, first consider the existence of a square-maximal (d,n)-string with singletons: by Lemma 10, $\sigma_d(n)=\sigma_{d-1}(n-1)$. Then consider the existence of a singleton-free square-maximal string x not in $\mathcal{U}_d(n)$:

- (i) either x does not have an s-cover, in which case by Lemma 9, $\sigma_d(n) = \sigma_d(n-1)$;
- (ii) or x has an s-cover with two consecutive adjacent squares and by Lemma 8, $\sigma_d(n) \le \sigma_{d_1}(n_1)+\sigma_{d_2}(n_2)$ for some $2 \le d_1, d_2 \le d$ and some $n_1+n_2=n$, and so $\sigma_d(n) \le \sigma_d^-(n)$;
- (iii) or x has a special s-cover, but is not dense and by Lemma 5, $\sigma_d(n) \le \sigma_d^-(n)$.

7. Recursive computation of $\sigma_d(2d)$

To compute the values on the main diagonal we can use s-covers satisfying additional necessary parity condition. The s-cover $\{S_i=(s_i,e_i,p_i): 1\le i\le m\}$ of $x=x[1..n]$ satisfies the *parity condition* if for $1\le i<m$, $\mathcal{A}(x[1..e_i]) \cap \mathcal{A}(x[s_{i+1}..n]) \subseteq \mathcal{A}(x[s_{i+1}..e_i])$.

Lemma 11. *The singleton-free part of a square-maximal $(d, 2d)$-string x with all its singletons at the end has an s-cover satisfying the parity condition.*

Proof. We can assume that x has $0 \leq v \leq d-2$ singletons, all at the end. Let $k(x)$ be the core vector of x. Suppose the singleton-free part $x[1..\, 2d-v]$ does not have an s-cover, then there exist some $1 \leq i_0 \leq 2d-v$ such that $k_{i_0}(x) = 0$. Remove $x[i_0]$ to form a $(d, 2d-1)$-string y. Therefore, $\sigma_d(2d) = \boldsymbol{s}(x) \leq \boldsymbol{s}(y) \leq \sigma_d(2d-1) = \sigma_{d-1}(2d-2)$, a contradiction. So $x[1..\, 2d-v]$ has an s-cover $\{S_i : 1 \leq i \leq m\}$. Let us assume that the s-cover does not satisfy the parity condition. Then either

(*i*) $\bigvee_{1\leq i\leq t} S_i$ and $\bigvee_{t<i\leq m} S_i$ for some $1 \leq t \leq m$ are adjacent and their respective alphabets have at least one symbol in common, say c. If we replace c in $\bigvee_{1\leq i\leq t} S_i$ by a new symbol $\hat{\boldsymbol{c}} \notin \mathcal{A}(x)$, we get a new $(d+1, 2d)$-string y so that $\boldsymbol{s}(y) \geq \boldsymbol{s}(x)$. Thus $\sigma_d(2d) = \boldsymbol{s}(x) \leq \boldsymbol{s}(y) \leq \sigma_{d+1}(2d) = \sigma_d(2d-1) = \sigma_{d-1}(2d-2)$, a contradiction, or

(*ii*) $\bigvee_{1\leq i\leq t} S_i$ and $\bigvee_{t<i\leq m} S_i$ for some $1 \leq t \leq m$ are overlapping, and there is a symbol c occurring in $\bigvee_{1\leq i\leq t} S_i$ and in $\bigvee_{t<i\leq m} S_i$, but not in the overlap $S_t \cap S_{t+1}$. If we replace c in $\bigvee_{1\leq i\leq t} S_i$ by a new symbol $\hat{\boldsymbol{c}} \notin \mathcal{A}(x)$, we get a new $(d+1, 2d)$-string y so that $\boldsymbol{s}(y) \geq \boldsymbol{s}(x)$. Thus $\sigma_d(2d) = \boldsymbol{s}(x) \leq \boldsymbol{s}(y) \leq \sigma_{d+1}(2d) = \sigma_d(2d-1) = \sigma_{d-1}(2d-2)$, a contradiction.

□

With additional assumptions, Lemma 11 can be strengthened to exclude consecutive adjacent squares from the s-cover of a square-maximal $(d, 2d)$-string.

Lemma 12. *Let $\sigma_{d'}(2d') = d'$ for $d' < d$. Either $\sigma_d(2d) = d$ or for every square-maximal $(d, 2d)$-string x with v singletons all at the end, $0 \leq v \leq d-2$, its singleton-free part $x[1..\, 2d-v]$ has an s-cover satisfying the parity condition and which has no consecutive adjacent squares.*

Proof. The existence of the s-cover $\{S_i \mid 1 \leq i \leq m\}$ of $x[1..\, 2d-v]$ satisfying the parity condition follows from Lemma 11. We need to prove that either $\sigma_d(2d) = d$ or there are no adjacent squares in the s-cover. Since $\sigma_{d'}(2d') = d'$ for $d' < d$, $\sigma_{d'}(n') \leq n' - d'$ for $n' - d' < d$. Let us assume that the s-cover of x has two adjacent squares S_t and S_{t+1}. Let $x_1 = \bigvee_{1\leq i\leq t} S_i$ and

let $x_2 = \bigvee_{t<i\le m} S_i$. Then $\boldsymbol{s}(x) \le \boldsymbol{s}(x_1) + \boldsymbol{s}(x_2)$ where x_1 and x_2 are, respectively, a (d_1, n_1)- and a (d_2, n_2)-string with $n_1+n_2 = 2d-v$ and $d_1+d_2 \ge d-v$. Since the s-cover satisfies the parity condition, $\mathcal{A}(x_1)$ and $\mathcal{A}(x_2)$ are disjoint and hence $d_1 + d_2 = d - v$. Therefore $(n_1 - d_1) + (n_2 - d_2) = d$. Since both x_1 and x_2 are singleton-free, $n_1 - d_1 > 0$ and $n_2 - d_2 > 0$. Hence $n_1 - d_1 < d$ and $n_2 - d_2 < d$, and therefore $\sigma_d(2d) = \boldsymbol{s}(x) \le \boldsymbol{s}(x_1) + \boldsymbol{s}(x_2) \le \sigma_{d_1}(n_1) + \sigma_{d_2}(n_2) \le (n_1 - d_1) + (n_2 - d_2) = d$. □

Since the number of distinct squares in a singleton-free $(d, 2d)$-string is at most d, we do not need to consider singleton-free strings. Moving a singleton to the end of a string does not decrease the number of distinct squares, therefore we shall only consider $(d, 2d)$-strings that have singletons at the end. We can set $\sigma_d^-(2d) = \sigma_{d-1}(2d-2)+1$ and thus consider only the strings that have the non-singleton part dense. By Lemma 12, we need only to consider strings whose s-covers of the non-singleton part satisfy the parity condition with no consecutive adjacent squares. Moreover, the number of singletons must be at least $\lceil \frac{2d}{3} \rceil$, see [7]. Let $\mathcal{T}_v$ denote the set of all singleton-free $\sigma_d^-(2d)$-dense $(d - \lceil \frac{2d}{3} \rceil, 2d - \lceil \frac{2d}{3} \rceil)$-strings admitting an s-cover satisfying the parity condition with no consecutive adjacent squares. Thus, we set

$$\sigma_d(2d) = \max\, \{d,\ \max_{x \in \mathcal{T}_v} \boldsymbol{s}(x)\}.$$

8. Additional properties of $\boldsymbol{\sigma_d(n)}$

Though key properties of $\sigma_d(n)$ were presented in [7], we present some additional properties concerning the gaps between consecutive values in the $(d, n-d)$ table where the value of $\sigma_d(n)$ is the entry on the d-th row and the $(n-d)$-th column. Lemma 13, respectively Lemma 14, shows that the difference between any two consecutive entries along a row, respectively between any two consecutive entries on the main diagonal, in the $(d, n-d)$ table is bounded by 2.

Lemma 13. *For $2 \le d \le n$, $\sigma_d(n+1) - \sigma_d(n) \le 2$.*

Proof. Let $(d, n+1)$-string $x = x[1..\, n+1]$ be square-maximal, then $\boldsymbol{s}(x) = \sigma_d(n+1)$. We can assume that the first symbol of x is not a singleton as otherwise we can move all singletons from the beginning of x to the end of x without destroying any square type. Let $y = x[2..\, n+1]$. Then y is a

(d, n)-string and $\boldsymbol{s}(y) \leq \sigma_d(n)$. By Fraenkel and Simpson [10], there are at most two rightmost occurrences of squares starting at the same position in a string. In other words, the removal of $x[1]$ destroyed at most two square types. That is, $\boldsymbol{s}(x) - 2 \leq \boldsymbol{s}(y)$. Therefore, $\sigma_d(n+1) - 2 \leq \boldsymbol{s}(y) \leq \sigma_d(n)$, implying $\sigma_d(n+1) - \sigma_d(n) \leq 2$. □

Lemma 14. *For $2 \leq d$, $\sigma_{d+1}(2d+2) - \sigma_d(2d) \leq 2$.*

Proof. By Lemma 13, $\sigma_{d+1}(2d+2) - \sigma_{d+1}(2d+1) \leq 2$. The entries under and on the main diagonal along a column are constant, see [7]; that is, $\sigma_{d+1}(2d+1) = \sigma_d(2d)$. Therefore, $\sigma_{d+1}(2d+2) - \sigma_d(2d) \leq 2$. □

Lemma 15. *For $d \geq 2$, if there is a square-maximal singleton-free $(d, 2d+1)$-string x, then there exists a square-maximal $(d, 2d+1)$-string y of the form $y = aaabbccdd....$*

Proof. Since x contains no singletons, then x contains exactly $d-1$ pairs and 1 triple. To prove there exists a square-maximal string in the form that all pairs consist of adjacent symbols and the triple also consists of adjacent symbols, we need to show the non-adjacent symbols can be moved together without reducing the number of distinct squares. Let us suppose that there is a non-adjacent pair of $\boldsymbol{c}$'s in x.

(i) If the $\boldsymbol{c}$'s did not occur in any square, then we could move both $\boldsymbol{c}$'s to the end of the string without destroying any square type. Moreover, we would gain a new square $\boldsymbol{cc}$, contradicting the square-maximality of x.
(ii) If the $\boldsymbol{c}$'s occur in exactly one square $u\boldsymbol{c}vu\boldsymbol{c}v$, where u and v are some strings, we can move both $\boldsymbol{c}$'s to the end of x to form a new string y. The new squares created by this move are $uvuv$ and $\boldsymbol{cc}$ while the old square $u\boldsymbol{c}vu\boldsymbol{c}v$ was destroyed. If $uvuv$ did not exist in any other part of x, then $\boldsymbol{s}(y) > \boldsymbol{s}(x)$ which contradicts the square-maximality of x; thus $uvuv$ already existed in some other part of x, so we lost the square $u\boldsymbol{c}vu\boldsymbol{c}v$, but gained $\boldsymbol{cc}$, so $\boldsymbol{s}(y) = \boldsymbol{s}(x)$.
(iii) If the $\boldsymbol{c}$'s occur in more than one square, these squares must form a non-trivial run, i.e. a run with a non-empty tail. Since there is only one symbol $\boldsymbol{t}$ occurring in x 3 times, the only form of such a non-trivial run can be $\boldsymbol{t}u\boldsymbol{c}v\boldsymbol{t}u\boldsymbol{c}v\boldsymbol{t}$. If $u = v = \varepsilon$, then the run is $\boldsymbol{tctct}$ containing two distinct

squares $\boldsymbol{tctc}$ and $\boldsymbol{ctct}$. We can change it to $\boldsymbol{tttcc}$, destroying the two squares $\boldsymbol{tctc}$ and $\boldsymbol{ctct}$, but gaining two new squares $\boldsymbol{tt}$ and $\boldsymbol{cc}$. If either $u \neq \varepsilon$ or $v \neq \varepsilon$, then by moving both $\boldsymbol{c}$'s to the end of x, we destroy the two distinct squares $\boldsymbol{t}u\boldsymbol{c}v\boldsymbol{t}u\boldsymbol{c}v$ and $u\boldsymbol{c}v\boldsymbol{t}u\boldsymbol{c}v\boldsymbol{t}$, but gain three new squares $\boldsymbol{t}uv\boldsymbol{t}uv$, $uv\boldsymbol{t}uv\boldsymbol{t}$, and $\boldsymbol{cc}$. Note that neither $\boldsymbol{t}uv\boldsymbol{t}uv$ nor $uv\boldsymbol{t}uv\boldsymbol{t}$ can exist anywhere else in x for the lack of $\boldsymbol{t}$'s. Thus we have more distinct squares than x, which contradicts the maximality of x.
Since we can move safely all pairs together to the end of x, the symbols of the triple will end up also adjacent at the beginning of the string. □

Lemma 16 shows that the two entries of the $(d, n-d)$ table in the same column just above the main diagonal must be identical.

Lemma 16. *For* $3 \leq d$, $\sigma_d(2d+1) = \sigma_{d-1}(2d)$.

Proof. We prove it by induction. Let (H_d) be the statement that $\sigma_d(2d+1) = \sigma_{d-1}(2d)$. (H_d) for $2 \leq d \leq 10$ is true from the values in the $(d, n-d)$ table computed so far, see [7]. Thus let us assume that H_1 through H_{d-1} are true, and let us prove that (H_d) is true. Let $(d, 2d+1)$-string x be square-maximal. If x contains a singleton, remove it to form a new $(d-1, 2d)$-string y. Then $\sigma_d(2d+1) = \boldsymbol{s}(x) \leq \boldsymbol{s}(y) \leq \sigma_{d-1}(2d)$ and since $\sigma_d(2d+1) \geq \sigma_{d-1}(2d)$, see [7], thus $\sigma_d(2d+1) = \sigma_{d-1}(2d)$. If x contains no singletons, by Lemma 15 we can assume that it has the form $aaabbccdd...$. Remove a pair from z forming a new $(d-1, 2d-1)$-string y. Then $\sigma_d(2d+1) - 1 = \boldsymbol{s}(x) - 1 = \boldsymbol{s}(y) \leq \sigma_{d-1}(2d-1)$ and since $\sigma_d(2d+1) - 1 \geq \sigma_{d-1}(2d-1)$ by [7], therefore $\sigma_d(2d+1) = \sigma_{d-1}(2d-1) + 1$. Since H_{d-1}, $\sigma_{d-1}(2d) \geq \sigma_{d-2}(2d-2) + 1$ and $\sigma_d(2d+1) \geq \sigma_{d-1}(2d)$ according to [7], hence $\sigma_d(2d+1) = \sigma_{d-1}(2d)$. □

Corollary 17 demonstrates that the difference between any two consecutive entries on the two diagonals immediately above the main diagonal of the $(d, n-d)$ table is bounded by 2.

Corollary 17. *For* $3 \leq d$, $\sigma_d(2d+1) - \sigma_{d-1}(2d-1) \leq 2$ *and* $\sigma_d(2d+2) - \sigma_{d-1}(2d) \leq 2$.

Proof. By Lemma 13, $\sigma_{d-1}(2d) - \sigma_{d-1}(2d-1) \leq 2$, and by Lemma 16, $\sigma_{d-1}(2d) = \sigma_d(2d+1)$. Therefore, $\sigma_d(2d+1) - \sigma_{d-1}(2d-1) \leq 2$. Similarly, $\sigma_d(2d+2) - \sigma_d(2d+1) \leq 2$ by Lemma 13, and $\sigma_d(2d+1) = \sigma_{d-1}(2d)$ by

Lemma 16. Therefore, $\sigma_d(2d+2) - \sigma_{d-1}(2d) \leq 2$. □

Remark 18. *Fraenkel and Simpson [10] gave a universal upper bound of* $2n$ *for distinct squares. Ilie [13] provided an asymptomatic bound of* $2n - \Theta(\log n)$. *Deza, Franek, and Thierry in [8] gave an improved universal upper bound of* $\lfloor \frac{11n}{6} \rfloor$. *In the following we provide some local bounds based on the computational results and the observations that for* $2 \leq d \leq n$ *and* $n \geq d_0+2$, $\sigma_d(n) \leq 2n - d_0 - 2d$, *where* d_0 *is the maximum* d *such that* $\sigma_d(2d) = d$ *is known. The current value of* $d_0 = 24$. *If, for a given* d, n_d *is the largest* n *such that* $\sigma_d(n_d)$ *is known, then*

$$\sigma_d(n) \leq \begin{cases} n-d & \text{if } n \leq n_d \\ 2n - (2n_d - \sigma_d(n_d)) & \text{if } n_d < n < 6(2n_d - \sigma_d(n_d)) \\ \lfloor \frac{11n}{6} \rfloor & \text{if } n \geq 6(2n_d - \sigma_d(n_d)) \end{cases}$$

Since $n_2 = 70$ *and* $\sigma_2(70) = 55$, *then* $\sigma_2(n) \leq \begin{cases} n-2 & \text{if } n \leq 70 \\ 2n-75 & \text{if } 70 < n < 510 \\ \lfloor \frac{11n}{6} \rfloor & \text{if } n \geq 510 \end{cases}$

Similarly slightly improved local bounds for $\sigma_3(n), \ldots, \sigma_{20}(n)$ *can be derived.*

Proof of the observation: *By Lemma 14,* $\sigma_d(n) \leq d_0+2k$, *where* $n-d = d_0+k$ *and* $k \geq 1$. *Thus* $\sigma_d(d_0+k+d) \leq d_0+2k = 2(d_0+k+d)-d_0-2d$. *Therefore,* $\sigma_d(n) \leq 2n - d_0 - 2d$ *for* $n \geq d_0 + 2$. □

9. Computational Results

We implemented the described algorithms in C++, and ran the programs in parallel on the Shared Hierarchical Academic Research Computing Network (SHARCNET) computer cluster. We were able to compute all $\sigma_2(n)$ values for $n \leq 70$ in a matter of hours. The higher the d, the longer the computations: currently we have these maximal values computed: $\sigma_2(70) = 55$, $\sigma_3(45) = 34$, $\sigma_4(38) = 27$, $\sigma_5(37) = 26$, $\sigma_6(35) = 24$, $\sigma_7(37) = 25$, $\sigma_8(29) = 18$, $\sigma_9(31) = 19$, $\sigma_{10}(33) = 20$, $\sigma_{11}(35) = 21$, $\sigma_{12}(31) = 17$, $\sigma_{13}(33) = 18$, $\sigma_{14}(35) = 19$, $\sigma_{15}(37) = 20$, $\sigma_{16}(37) = 19$, $\sigma_{17}(37) = 18$, $\sigma_{18}(38) = 19$, $\sigma_{19}(40) = 20$, and $\sigma_{20}(41) = 20$. The results and sample square-maximal strings may be found at [6]. Whenever the computation required determining the number of distinct primitively rooted squares in a

concrete string, a C++ implementation of the Franek, Jiang, and Weng's algorithm [12] was used. The values of interest include three consecutive equal values: $\sigma_2(31) = \sigma_2(32) = \sigma_2(33)$, and unexpected pairs (d, n) satisfying $\sigma_{d+1}(n+2) - \sigma_d(n) > 1$ such as (2,33) and (2,34). To show that among all strings of length 33, the maximum number of distinct primitively rooted squares can not be achieved by a non-ternary string, we reproduce in Table 2 a fragment of the $(d, n-d)$ table from [6] where the entries for strings of length 33 are in bold. All known values for strings of length 33 are at most 23 except $\sigma_3(33) = 24$. The 3 undetermined entries $\sigma_{12}(33), \sigma_9(33)$, and $\sigma_8(33)$ are at most 23 using the following observations: $\sigma_{12}(33) \leq \sigma_{14}(35) = 19$, $\sigma_9(33) \leq \sigma_{11}(35) = 21$, and $\sigma_8(33) \leq \sigma_8(32) + 2 \leq \sigma_9(33) + 2 = 23$.

		$n-d$															
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
d	2	12	13	13	14	15	16	17	18	19	20	20	21	22	23	23	**23**
	3	13	13	14	14	15	16	17	18	19	20	21	21	22	23	**24**	24
	4	13	14	14	15	15	16	17	18	19	20	21	22	22	**23**	24	25
	5	13	14	15	15	16	16	17	18	19	20	21	22	**23**	23	24	25
	6	13	14	15	16	16	17	17	18	19	20	21	**22**	23	24	?	?
	7	13	14	15	16	17	17	18	18	19	20	**21**	22	23	24	25	?
	8	13	14	15	16	17	18	?	?	?	**$\sigma_8(33)$**	?	?	?	?	?	?
	9	14	14	15	16	17	18	19	?	**$\sigma_9(33)$**	?	?	?	?	?	?	?
	10	14	15	15	16	17	18	19	**20**	?	?	?	?	?	?	?	?
	11	14	15	16	16	17	18	**19**	20	21	?	?	?	?	?	?	?
	12	14	15	16	17	?	**$\sigma_{12}(33)$**	?	?	?	?	?	?	?	?	?	?
	13	14	15	16	17	**18**	?	?	?	?	?	?	?	?	?	?	?
	14	15	15	16	**17**	18	19	?	?	?	?	?	?	?	?	?	?
	15	15	16	**16**	17	18	19	20	?	?	?	?	?	?	?	?	?
	16	16	**16**	17	17	18	19	?	?	?	?	?	?	?	?	?	?

Table 2: $(d, n-d)$ table for $\sigma_d(n)$ with $2 \leq d \leq 16$ and $16 \leq n - d \leq 31$ where the entries for strings of length 33 are in bold

Acknowledgments. The authors would like to thank the anonymous referees for valuable comments and suggestions which improved the quality of the paper. This work was supported by grants from the Natural Sciences and Engineering Research Council of Canada Discovery Grant program (Franek: RGPIN25112-2012, Deza: RGPIN-2015-06163), by the Digiteo Chair C&O program (Deza), and by the Canada Research Chairs program (Deza: CRC 950-213642), and made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (`http://www.sharcnet.ca/`).

[1] A. Baker, A. Deza, and F. Franek. On the structure of run-maximal strings. *Journal of Discrete Algorithms*, 14:10–14, 2012.

[2] A. Baker, A. Deza, and F. Franek. A computational framework for determining run-maximal strings. *Journal of Discrete Algorithms*, 20:43 – 50, 2013.

[3] H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta. The "Runs" Theorem. *arXiv:1406.0263v6*, 2015.

[4] A. Deza and F. Franek. A d-step approach to the maximum number of distinct squares and runs in strings. *Discrete Applied Mathematics*, 163:268–274, 2014.

[5] A. Deza and F. Franek. Bannai et al. method proves the d-step conjecture for strings. AdvOL-Report 2015/1, McMaster University, 2015.

[6] A. Deza, F. Franek, and M. Jiang. Square-maximal strings. `http://optlab.mcmaster.ca/~jiangm5/research/square.html`.

[7] A. Deza, F. Franek, and M. Jiang. A d-step approach for distinct squares in strings. In *Combinatorial Pattern Matching - 22nd Annual Symposium, CPM 2011, Palermo, Italy, June 27-29, 2011. Proceedings*, pages 77–89, 2011.

[8] A. Deza, F. Franek, and A. Thierry. How many double squares can a string contain? *Discrete Applied Mathematics*, 180:52–69, 2015.

[9] J. Fischer, S. Holub, T. I, and M. Lewenstein. Beyond the runs theorem. In *String Processing and Information Retrieval - 22nd International Symposium, SPIRE 2015, London, UK, September 1-4, 2015, Proceedings*, pages 277–286, 2015.

[10] A. S. Fraenkel and J. Simpson. How many squares can a string contain? *Journal of Combinatorial Theory Series A*, 82:112–120, 1998.

[11] F. Franek and J. Holub. A different proof of Crochemore-Ilie lemma concerning microruns. In *London Algorithmics 2008: Theory and Practice*, pages 1–9. College Publications, London, UK, 2009.

[12] F. Franek, M. Jiang, and C. Weng. An improved version of the runs algorithm based on Crochemore's partitioning algorithm. In *Proceedings of Prague Stringology Conference 2011*, pages 98–105. Prague, Czech Republic, 2011.

[13] L. Ilie. A note on the number of squares in a word. *Theoretical Computer Science*, 380:373–376, 2007.