Journal of Discrete Algorithms 20 (2013) 43–50

Contents lists available at SciVerse ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

A computational framework for determining run-maximal strings

Andrew Baker, Antoine Deza [*], Frantisek Franek

Department of Computing and Software, McMaster University, 280 Main St. West, Hamilton, Ontario, Canada

ARTICLE INFO

Article history:
Available online 28 December 2012

Keywords:
Strings
Runs
R-covers
Run-maximality

ABSTRACT

We investigate the function $\rho_d(n) = \max\{\mathbf{r}(x) \mid x \text{ is a } (d,n)\text{-string}\}$, where $\mathbf{r}(x)$ denotes the number of runs in a string x and (d,n)-string denotes a string of length n with exactly d distinct symbols. The notion of an r-cover is presented and discussed with emphasis on the recursive computational determination of $\rho_d(n)$. This notion is used as a key element of a computational framework for an efficient computation of the maximum number of runs. In particular, we were able to determine all previously known $\rho_2(n)$ values for $n \leqslant 60$ in a matter of hours, confirming the results reported by Kolpakov and Kucherov, and were able to extend the computations up to and including $n = 74$. Noticeably, these computations reveal the unexpected existence of a binary run-maximal string of length 66 containing *aaaa*.

1. Introduction

In [2] the notion of an r-cover was introduced as a means to represent the distribution of the runs in a string and thus describe the structure of run-maximal strings. The straightforward assertion from [2] that a run-maximal string has an r-cover – except possibly a single weak point – holds only when the size of the alphabet is not kept fixed. However, the approach can be adapted inductively to handle situations with fixed alphabets and can be used to speed up computations of the maximum number of runs.

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 $e = s + 2p - 1$. Similarly, we encode a run as a triple (s, e, p). It is clear from the context whether a triple (s, e, p) encodes a square or a run. Note that the exponent of such a run equals $\lfloor \frac{e-s+1}{p} \rfloor$ and the tail of the run equals the remainder of the division of $(e - s + 1)$ by p. The *leading square* of a run (s, e, p) refers to the square $(s, s + 2p - 1, p)$. The *trailing square* of a run (s, e, p) refers to the square $(e - 2p + 1, e, p)$.

The *join* $x[i_1 \mathinner{..} i_k] \cup x[j_1 \mathinner{..} j_m]$ of two substrings of a string $x = x[1 \mathinner{..} n]$ is defined if $i_1 \leqslant j_1 \leqslant i_k + 1$ and then $x[i_1 \mathinner{..} i_k] \cup x[j_1 \mathinner{..} j_m] = x[i_1 \mathinner{..} \max\{i_k, j_m\}]$, or if $j_1 \leqslant i_1 \leqslant j_m + 1$ and then $x[i_1 \mathinner{..} i_k] \cup x[j_1 \mathinner{..} j_m] = x[j_1 \mathinner{..} \max\{i_k, j_m\}]$. Simply put, the join is defined when the two substrings are either adjacent or overlap. For two encodings (s_1, e_1, p_1) and (s_2, e_2, p_2) of squares in a string x, the join $(s_1, e_1, p_1) \cup (s_2, e_2, p_2)$ represents the join of $x[s_1 \mathinner{..} e_1] \cup x[s_2 \mathinner{..} e_2]$. The alphabet of x is denoted by $\mathcal{A}(x)$, a (d, n)-string refers to a string of length n with exactly d distinct symbols, $\mathbf{r}(x)$ denotes the number of runs in a string x, and $\rho_d(n)$ refers to the maximum number of runs over all (d, n)-strings, i.e. $\rho_d(n) = \max\{\mathbf{r}(x) \mid x \text{ is a } (d,n)\text{-string}\}$. The number of distinct symbols of a string x is denoted as $d(x)$. A *singleton* is a symbol which occurs exactly once in the string under consideration, so a singleton-free string is a string in which each symbol occurs at least twice. A square (s, e, p) is *left-shiftable* if $x[s - 1]$ is defined $(s > 1)$, and $x[s - 1] = x[s + p - 1]$.

[*] Corresponding author.
E-mail address: deza@mcmaster.ca (A. Deza).

1570-8667/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jda.2012.12.004

Similarly, a square is *right-shiftable* if $x[e+1]$ is defined $(e<n)$ and $x[e+1]=x[s]$. In other words, a square (s,e,p) is left-shiftable exactly when $(s-1,e-1,p)$ is also a square, and is right-shiftable exactly when $(s+1,e+1,p)$ is also a square. To simplify the notation, for the empty string ε we set $\mathbf{r}(\varepsilon)=0$ and $\rho_d(0)=0$.

Considering $\rho(n)$ the maximum number of runs over all strings of length n, i.e. $\rho(n)=\max\{\rho_d(n)\colon\ 1\leqslant d\leqslant n\}$, the investigation of the asymptotic behavior of $\rho(n)/n$ has provided a rich line of research. See [4,8] and references therein for more details and additional results and approaches.

2. Computational approach to runs

The computational framework for determining $\rho_d(n)$ presented in subsequent sections is based on the following approach: We first compute a lower bound of $\rho_d(n)$, denoted as $\rho_d^-(n)$. Then it is enough to restrict our search to the (d,n)-strings potentially satisfying $\mathbf{r}(x)>\rho_d^-(n)$, thus significantly reducing the search space. This section introduces necessary conditions guaranteeing that for a string x, $\mathbf{r}(x)>\rho_d^-(n)$. We show that for a string x to potentially satisfy $\mathbf{r}(x)>\rho_d^-(n)$, it must have an *r-cover* and be $\rho_d^-(n)$*-dense*. Only the r-covered strings are generated and the ones not satisfying the $\rho_d^-(n)$-density are eliminated at the earliest possible stage.

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

(1) none of the S_i's, $1\leqslant i\leqslant m$ is left-shiftable;
(2) $s_i<s_{i+1}\leqslant e_i+1<e_{i+1}+1$ for any $1\leqslant i<m$, i.e. two consecutive squares are either adjacent or overlap without one containing the other;
(3) $\bigcup_{1\leqslant i\leqslant m}S_i=x$;
(4) for any run $R=(s,e,p)$ of x there is an S_i with $1\leqslant i\leqslant n$ containing the leading square of the run R.

A string which has an r-cover is referred to as *r-covered*.

An r-cover with no adjacent squares is referred to as *overlapping r-cover*.

See Fig. 1 for an illustration of an overlapping r-cover.

Lemma 2. *The r-cover of an r-covered string is unique.*

Proof. Let us assume that we have two different r-covers of x, $\{S_i\mid 1\leqslant i\leqslant m\}$ and $\{S'_j\mid 1\leqslant j\leqslant k\}$. We shall prove by induction that they are identical. By Definition 1(4), S_1 is a substring of S'_1 and, by the same argument, S'_1 is a substring of S_1, and thus $S_1=S'_1$. Let the induction hypothesis be $S_i=S'_i$ for $1\leqslant i\leqslant t$. If $\bigcup_{1\leqslant i\leqslant t}S_i=x$, we have $t=m=k$ and we are done. Otherwise consider S_{t+1}. By Definition 1(4), there is S'_v so that S_{t+1} is a substring of S'_v and $v>t$. We need to show that $v=t+1$. If not, then S'_{t+1} is a substring of $\bigcup_{1\leqslant i\leqslant t+1}S_i$ as otherwise S'_{t+1} would contain S_{t+1}, contradicting $v\neq t+1$. Since S'_{t+1} is not a substring of $\bigcup_{1\leqslant i\leqslant t}S_i$, then S'_{t+1} is a substring of S_{t+1}, which in turn is a substring of S'_v, a contradiction. Therefore, S_{t+1} is a substring of S'_{t+1}. Similarly, S'_{t+1} i a substring of S_{t+1} and so $S_{t+1}=S'_{t+1}$, which completes the induction. □

Lemma 3. *Any r-covered string is singleton-free.*

Proof. Let $\{S_j\mid 1\leqslant j\leqslant m\}$ be the r-cover of $x=x[1\mathinner{..}n]$. For any $1\leqslant i\leqslant n$, $x[i]\in S_t$ for some t by Definition 1(3). Since S_t is a square, the symbol $x[i]$ occurs in x at least twice. □

Before defining a $\rho_d^-(n)$*-dense string*, we recall the notion of a *core* of a run introduced in [5]: for a run (s,e,p), its core is the intersection of the set of indices of its leading square $(s,s+2p-1,p)$ and the set of indices of its trailing square $(e-2p+1,e,p)$.

Definition 4.

(a) Let $k_i(x)$ be the number of cores in x containing the position i. Given a (d,n)-string x, the vector $k(x)=(k_1(x),\ldots,k_n(x))$ is referred to as the *core vector* of x.
(b) A singleton-free (d,n)-string x is $\rho_d^-(n)$*-dense*, if its core vector $k(x)$ satisfies $k_i(x)>\rho_d^-(n)-\mathbf{r}(x[1\mathinner{..}i-1])-m_i$ for $i=1\mathinner{..}n$, where $m_i=\max\{\rho_{d_2}(n-i)\colon\ d-d_1\leqslant d_2\leqslant\min(n-i,d)\}$ and $d_1=d(x[1\mathinner{..}i-1])$.

Lemma 5. *If a (d,n)-string x is not $\rho_d^-(n)$-dense, then $\mathbf{r}(x)\leqslant\rho_d^-(n)$.*

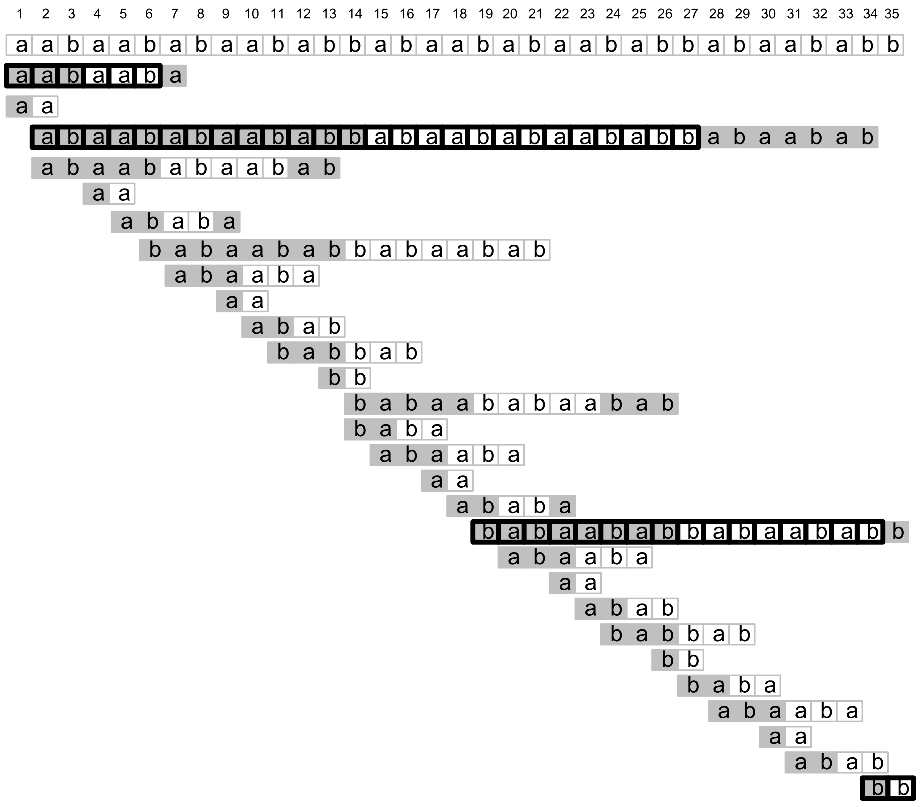

Fig. 1. An illustration of all the runs in a string, with the squares of the r-cover indicated in bold.

Proof. Clearly, for any string x, $\boldsymbol{r}(x) \leqslant \boldsymbol{r}(x[1\ ..\ i-1]) + \boldsymbol{r}(x[i+1\ ..\ n]) + k_i(x)$ for all i's. Note that in most situations $\boldsymbol{r}(x) = \boldsymbol{r}(x[1\ ..\ i-1]) + \boldsymbol{r}(x[i+1\ ..\ n]) + k_i(x)$, except when the core of some run containing i is empty – such run is split into two runs: one in $x[1\ ..\ i-1]$ and the other in $x[i+1\ ..\ n]$. If x is not $\rho_d^-(n)$-dense, then for some i_0, $k_{i_0}(x) \leqslant \rho_d^-(n) - \boldsymbol{r}(x[1\ ..\ i_0-1]) - m_{i_0}$. Since $\boldsymbol{r}(x) \leqslant \boldsymbol{r}(x[1\ ..\ i_0-1]) + \boldsymbol{r}(x[i_0+1\ ..\ n]) + k_{i_0}(x) \leqslant \boldsymbol{r}(x[1\ ..\ i_0-1]) + m_{i_0} + k_{i_0}(x)$, then $\boldsymbol{r}(x) \leqslant \boldsymbol{r}(x[1\ ..\ i_0-1]) + m_{i_0} + \rho_d^-(n) - \boldsymbol{r}(x[1\ ..\ i_0-1]) - m_{i_0} = \rho_d^-(n)$. □

Lemma 6. *If the core vector of a (d,n)-string has non-zero entries, then the string has an r-cover.*

Proof. We build an r-cover by induction: Since the $k_1(x) \neq 0$, position 1 is in at least one core, hence there must be at least one run starting at position 1. Among all runs starting at position 1, set S_1 to the leading square of the run with the largest period. Let the inductive hypothesis be that we have built the r-cover up to $i \leqslant t$: $\{S_i = (s_i, e_i, p_i) \mid 1 \leqslant t\}$. If $\bigcup_{1\leqslant i\leqslant t} S_i = x$, we are done. Otherwise $\bigcup_{1\leqslant i\leqslant t} S_i = x[1\ ..\ e_t]$. Since $k_{e_t+1}(x) \neq 0$, there is at least one run (s,e,p) in x such that $s \leqslant e_t + 1 \leqslant s + 2p - 1$. From all such runs chose the set of runs with the leftmost starting position, and among them choose the one with the largest period, and set S_{t+1} to the leading square of the chosen run. It is straightforward to verify that all the conditions of Definition 1 are satisfied and that we have built an r-cover of x. □

Lemma 6 yields a computationally efficient generalization allowing the determination of previously intractable values of $\rho_d(n)$. Namely, the generation of r-covered strings is computationally tractable as opposed to the generation of strings satisfying $k_i(x) \neq 0$. Note that being r-covered for a (d,n)-string x can be interpreted as a minor generalization of satisfying $k_i(x) \leqslant 2$ for $1 \leqslant i \leqslant n$.

Lemma 7 shows how $k(x)$ can be estimated from the partially generated r-cover of x. It is used in the following way: r-covered strings are generated and the ones with core vectors not meeting a certain threshold vector ought to be eliminated. Lemma 7 shows that the estimates for $k_i(x)$ are non-increasing with the increase of the partial r-cover.

Lemma 7. *Let $\{S_j = (s_j, e_j, p_j) \mid 1 \leqslant j \leqslant m\}$ be the r-cover of a (d,n)-string x and $1 \leqslant j_1 \leqslant j_2 \leqslant m$, then $k_i(x) = k_i(x[1\ ..\ e_m]) \leqslant k_i(x[1\ ..\ e_{j_2}]) \leqslant k_i(x[1\ ..\ e_{j_1}])$ for $1 \leqslant i \leqslant e_{j_1}$.*

Proof. Let R be a run in $x[1\,..\,e_{j_2}]$ containing i in its core. By Definition 1(4), the leading square of R is a substring of S_j for some $1 \leqslant j \leqslant j_2$. Since i lies in S_j, it follows that $j \leqslant j_1$, and so the part of R lying in $x[1\,..\,e_{j_1}]$ is a run in $x[1\,..\,e_{j_1}]$ with a non-empty core containing i, therefore, $k_i(x[1\,..\,e_{j_2}]) \leqslant k_i(x[1\,..\,e_{j_1}])$. □

Lemma 8. *If there is a run-maximal (d,n)-string x with an r-cover with a pair of adjacent squares, then $\rho_d(n) \leqslant \rho_{d_1}(n_1) + \rho_{d_2}(n_2)$ for some $2 \leqslant d_1, d_2 \leqslant d$ and some $n_1, n_2 \geqslant 0$ such that $n_1 + n_2 = n$.*

Proof. Let $\{S_i\colon\ 1 \leqslant i \leqslant m\}$ be the r-cover of x. Let $S_j = (s_j, e_j, p_j)$ and $S_{j+1} = (s_{j+1}, e_{j+1}, p_{j+1})$ be two adjacent squares of the r-cover, i.e. $s_{j+1} = e_j + 1$. Let $x_1 = \bigcup_{1 \leqslant i \leqslant j} S_i$ and $x_2 = \bigcup_{j < i \leqslant m} S_i$. Clearly $\rho_d(n) = \boldsymbol{r}(x) \leqslant \boldsymbol{r}(x_1) + \boldsymbol{r}(x_2) \leqslant \rho_{d(x_1)}(|x_1|) + \rho_{d(x_2)}(|x_2|)$. □

Lemma 9. *If a singleton-free run-maximal (d,n)-string x does not have an r-cover, then $\rho_d(n) \leqslant \rho_d(n_1) + \rho_d(n_2)$ for some $n_1, n_2 \geqslant 0$ such that $n_1 + n_2 = n - 1$.*

Proof. Since x does not have an r-cover, there is an $x[i]$ that is not in the core of any run. Consider substrings $x_1 = x[1\,..\,i-1]$ and $x_2 = x[i+1\,..\,n]$. We consider two cases: Case (a): If $\mathcal{A}(x) = \mathcal{A}(x_1) = \mathcal{A}(x_2)$, then $\rho_d(n) = \boldsymbol{r}(x) \leqslant \boldsymbol{r}(x_1) + \boldsymbol{r}(x_2) \leqslant \rho_d(|x_1|) + \rho_d(|x_2|)$. Case (b): If $\mathcal{A}(x_1) \neq \mathcal{A}(x_2)$, then without loss of generality, assume there is $\boldsymbol{c} \in \mathcal{A}(x_1) \setminus \mathcal{A}(x_2)$. Permute the alphabet of x_1 creating a new string $\widetilde{x_1}$, so that $\widetilde{x_1}[i-1] = \boldsymbol{c}$. Then $\widetilde{x_1}$ and x_2 can be concatenated into a string of length $n-1$ without merging any runs. Therefore, $\rho_d(n) = \boldsymbol{r}(x) \leqslant \boldsymbol{r}(\widetilde{x_1}) + \boldsymbol{r}(x_2) = \boldsymbol{r}(x_1) + \boldsymbol{r}(x_2) \leqslant \rho_d(n-1) \leqslant \rho_d(n)$, and so $\rho_d(n) = \rho_d(n-1)$. □

Lemma 10. *If a run-maximal (d,n)-string has a singleton, then either $\rho_d(n) = \rho_{d-1}(n-1)$ or $\rho_d(n) = \rho_d(n-1)$.*

Proof. For a given run-maximal (d,n)-string x there are three cases: Case (a): x has a singleton at the end or the beginning. If it is at the end, then $\rho_d(n) = \boldsymbol{r}(x) = \boldsymbol{r}(x[1\,..\,n-1]) \leqslant \rho_{d-1}(n-1)$ as $x[1\,..\,n-1]$ is a $(d-1, n-1)$-string. It follows that $\rho_d(n) = \rho_{d-1}(n-1)$. For a singleton at the beginning the proof is identical. Case (b): x has a singleton in the middle at a position j and the alphabets of the two parts are different, i.e. there is $\boldsymbol{c}$ so that either $\boldsymbol{c} \in \mathcal{A}(x[1\,..\,j-1]) \setminus \mathcal{A}(x[j+1\,..\,n])$ or $\boldsymbol{c} \in \mathcal{A}(x[j+1\,..\,n]) \setminus \mathcal{A}(x[1\,..\,j-1])$. If $\boldsymbol{c} \in \mathcal{A}(x[j+1\,..\,n]) \setminus \mathcal{A}(x[1\,..\,j-1])$, then create x_1 by permuting the alphabet of $x[j+1\,..\,n]$ so that $\boldsymbol{c}$ moves to the position $j+1$. This will not affect the number of runs and so $\boldsymbol{r}(x) = \boldsymbol{r}(x_1)$. Create x_2 by moving the singleton $x[j]$ to the end. Again, the number of runs will not be affected and so $\boldsymbol{r}(x_1) = \boldsymbol{r}(x_2)$. Then $y = x_2[1\,..\,n-1]$ is a $(d-1, n-1)$-string and $\rho_d(n) = \boldsymbol{r}(x) = \boldsymbol{r}(x_2) = \boldsymbol{r}(y) \leqslant \rho_{d-1}(n-1) \leqslant \rho_d(n)$. The argument is similar if $\boldsymbol{c} \in \mathcal{A}(x[1\,..\,j-1]) \setminus \mathcal{A}(x[j+1\,..\,n])$. Case (c): x has a singleton $\boldsymbol{c}$ in the middle at a position j and the alphabets of the two parts are the same; that is, $\mathcal{A}(x[j+1\,..\,n]) = \mathcal{A}(x[1\,..\,j-1]) = \mathcal{A}(x) - \{\boldsymbol{c}\}$. Replace all occurrences of $x[j+1]$ in $x[j+1\,..\,n]$ with the singleton $\boldsymbol{c}$, producing x_1. This will not affect any runs and so $\boldsymbol{r}(x) = \boldsymbol{r}(x_1)$. Moreover, $\mathcal{A}(x) = \mathcal{A}(x_1)$. Create a string x_2 by removing $x_1[j]$. Since no runs are merged, $\boldsymbol{r}(x_1) = \boldsymbol{r}(x_2)$. Since $\mathcal{A}(x) = \mathcal{A}(x_2)$, x_2 is a $(d, n-1)$-string and thus $\rho_d(n) = \boldsymbol{r}(x) = \boldsymbol{r}(x_2) \leqslant \rho_d(n-1) \leqslant \rho_d(n)$. □

Corollary 11. *If there is a run-maximal $(d, 2d)$-string with t singletons, then there is a run-maximal $(d, 2d)$-string with t singletons at the end.*

Proof. Consider the proof of Lemma 10. If x has a singleton at the beginning, case (a) of the proof of Lemma 10, it can be moved to the end without affecting the number of runs.

If x has a singleton in the middle and $\mathcal{A}(x[1\,..\,i-1]) \neq \mathcal{A}(x[i+1\,..\,2d])$, case (b) of the proof of Lemma 10, then we can make the transformation preserving the number of runs but changing the singleton to a multiply-occurring symbol while preserving any other singleton and its position. If x has a singleton in the middle and $\mathcal{A}(x[1\,..\,i-1]) = \mathcal{A}(x[i+1\,..\,2d])$, case (c) of the proof of Lemma 10, then $\rho_d(2d) = \rho_d(2d-1) = \rho_{d-1}(2d-2)$, which is impossible. Recall that $\rho_d(2d) = \rho_{n-d}(2n-2d)$ for $2 \leqslant d \leqslant n < 2d$, see [3], and that $\rho_d(n) > \rho_{d-1}(n-2)$ since appending two copies zz of a new symbol $z \notin \mathcal{A}(x)$ to the end of any string x increases the number of runs by 1. □

3. Heuristic for a lower bound $\rho_d^-(n)$

The higher the value of $\rho_d^-(n)$, the less computational effort must be spent on determining $\rho_d(n)$. For $d = 2$, generate $\mathcal{L}_2(n)$, the set of $(2,n)$-strings which: (a) are r-covered with overlapping r-cover, (b) are balanced over every prefix; that is, the frequencies of a's and b's differ by at most a constant selected according to an analysis of smaller binary run-maximal strings, (c) have a maximum period bounded by at most a predefined constant, and (d) contain no triples (aaa or bbb). We set the value of $\rho_2^-(n)$ to be:

$$\rho_2^-(n) = \max\Big\{\rho_2(n-1),\ \rho_2(n-2)+1,\ \max_{x \in \mathcal{L}_2(n)} \boldsymbol{r}(x)\Big\}.$$

Algorithm 1: Generating strings with overlapping r-cover.

```
begin Set the first square, $S_1$:
    $s_1 \leftarrow 1$
    for $p_1 \leftarrow 1 .. \lfloor \frac{n}{2} \rfloor$ do
        $e_1 \leftarrow 2p_1$
        foreach primitive string $g$ of length $p_1$ do
            $x[1 .. 2 * p_1 - 1] \leftarrow gg$
            addSquare(2)

begin addSquare($t$)
    for $s_t \leftarrow s_{t-1} + 1 .. e_t$ do
        for $e_t \leftarrow e_{t-1} + 1 .. n$ do
            if $e_t - s_t + 1$ is even then
                $p_t \leftarrow (e_t - s_t + 1)/2$
                if $s_t + p_t - 1 \leqslant e_{t-1}$ then
                    if the squaring of the generator locally coincides with the existing string then
                        square the generator
                        finishSquare($t$)
                        remove the square
                else
                    foreach completion of the generator do
                        square the generator
                        finishSquare($t$)
                        remove the square

begin finishSquare($t$)
    if $s_t - 1 \neq e_t$ then (the square is not left-shiftable)
        if $x[s_t .. e_t]$ is primitive then
            if no square $(s, e, p)$ so that $s_{t-1} \leqslant s \leqslant s_t \leqslant e_{t-1} \leqslant e \leqslant e_t$ and $(s, s + 2p - 1, p) \notin \{S_i \mid 1 \leqslant i \leqslant t\}$ then
                if $e_t = n$ then
                    if number of distinct symbols $= d$ then
                        output $x$
                else
                    addSquare($t + 1$)
```

This heuristic was found to be highly efficient when tested against the known run-maximal strings for $\rho_2(n)$: Franek and Smyth up to 34, and Kolpakov and Kucherov [7] up to 60. Note that $\rho_2^-(25) < \rho_2(25)$ since the only run-maximal $(2, 25)$-string contains a triple. For $d \geqslant 3$, we set $\rho_d^-(n) = \max\{\rho_{d-1}(n-1), \rho_{d-1}(n-2) + 1, \rho_d(n-1)\}$.

4. Generating (d, n)-strings with overlapping r-covers

This section describes the generation of all r-covered strings with overlapping r-covers. For a square uu, we refer to u as the *generator* of the square. We built a square by determining its generator. Starting from determining S_1, we recursively generate S_t given $\{S_i \mid 1 \leqslant i < t\}$. The pseudocode for this approach is given in Algorithm 1, while we describe below the steps.

Constructing S_1: The value of s_1 is set to 1 and all possible primitive strings u of length p_1 for $p_1 = 1 .. \lfloor \frac{n}{2} \rfloor$ are generated using a restricted growth string approach to avoid isomorphic duplicates with respect to the permutation of the alphabet. For every u generated, e_1 is set to $2|u| - 1$, $x[1 .. 2p_1 - 1]$ is set to uu, and S_1 is encoded by (s_1, e_1, p_1).

Constructing S_t: A partial overlapping r-cover $\{S_i \mid 1 \leqslant i < t\}$ has been built so that $x[1 .. e_{t-1}] = \bigcup_{1 \leqslant i < t} S_i$. Since every two consecutive squares in the r-cover being generated must overlap, it is enough to consider every pair s_t and e_t such that $s_{t-1} < s_t \leqslant e_{t-1} < e_t \leqslant n$ and $p_t = (e_t - s_t + 1)/2$ is an integer.

Case (a): $s_t + p_t - 1 \leqslant e_{t-1}$, then the generator of the square $S_t = (s_t, e_t, p_t)$ is already completely determined as it is a substring of $\bigcup_{1 \leqslant j < t} S_j$. If $s_t + p_t - 1 < e_{t-1}$, the entries of the square determined by the generator must coincide with $x[s_t .. e_{t-1}]$. Then the generator is tested for being primitive and not left-shiftable. Finally, it is tested whether extending $\bigcup_{1 \leqslant j < t} S_j$ by the square (s_t, e_t, p_t) does not introduce a so-called intermediate square, i.e. a non-left-shiftable square $(s, e, p) \notin \{S_j \mid 1 \leqslant j \leqslant t\}$ such that $s_{t-1} \leqslant s \leqslant s_t \leqslant e_{t-1} \leqslant e \leqslant e_t$ and $(s, s + 2p - 1, p)$. If the square $S_t = (s_t, e_t, p_t)$ passes all these tests, S_t it added to the r-cover and $x[1 .. e_{t-1}]$ is appended by the missing part of the second occurrence of the generator, and thus $x[1 .. e_t] = \bigcup_{1 \leqslant j \leqslant t} S_j$.

Case (b): Otherwise, all possible ways to extend the partial generator to the required length $s_t + p_t - 1$ are considered using a restricted growth string approach. Every extension is tested for being primitive and non-left-shiftable. Then the possible extension of the string is tested for introduction of intermediate squares, if none is found, the square $S_t = (s_t, e_t, p_t)$ is added to the r-cover and $x[1 \mathrel{..} e_{t-1}]$ is appended by the missing part of the generator and another copy of the generator, so $x[1 \mathrel{..} e_t] = \bigcup_{1 \leqslant j \leqslant t} S_j$. If $e_t = n$, the whole generated string is tested for having exactly d symbols, and if it has fewer than d, it is rejected.

A more restrictive generation may be required. For instance, when the lower bound $\rho_d^-(n)$ is being determined, see Section 3, the test of $\bigcup_{1 \leqslant j \leqslant t} S_j$ for being balanced and having no triples can be performed at each stage. Similarly, when strings with a required density are generated, see Section 5, the core vector of $\bigcup_{1 \leqslant j \leqslant t} S_j$ can be computed at each stage, and if it is insufficient, this branch of the generation process is terminated based on Lemma 7. When strings with overlapping r-covers satisfying parity condition are generated, see Section 5, the overlap between two consecutive squares of the partial r-cover can be tested at each stage for the parity condition, see Section 5.2.

5. Recursive computation of $\rho_d(n)$

5.1. General case

First, $\rho_d^-(n)$ is computed by the heuristic of Section 3. Then the following two inequalities (a) $\rho_d(n_1) + \rho_d(n_2) \leqslant \rho_d^-(n)$ for any $n_1 + n_2 = n - 1$, and (b) $\rho_{d_1}(n_1) + \rho_{d_2}(n_2) \leqslant \rho_d^-(n)$ for any $2 \leqslant d_1, d_2 \leqslant d$ and any $n_1 + n_2 = n$, are verified. Then $\mathcal{U}_d(n)$, the set of all $\rho_d^-(n)$-dense (d, n)-strings with overlapping r-covers is generated as described in Section 4. It follows that

$$\rho_d(n) = \max\left\{\rho_d^-(n), \max_{x \in \mathcal{U}_d(n)} \boldsymbol{r}(x)\right\}.$$

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

(a) either x does not have an r-cover and so $\rho_d(n) \leqslant \rho_d(n_1) + \rho_d(n_2)$ for some $n_1 + n_2 = n - 1$ by Lemma 9, and so $\rho_d(n) \leqslant \rho_d^-(n)$; or
(b) x has an r-cover with two adjacent squares and $\rho_d(n) \leqslant \rho_{d_1}(n_1) + \rho_{d_2}(n_2)$ for some $2 \leqslant d_1, d_2 \leqslant d$ and some $n_1 + n_2 = n$ by Lemma 8, thus $\rho_d(n) \leqslant \rho_d^-(n)$; or
(c) x has an overlapping r-cover, but is not $\rho_d^-(n)$-dense, in which case $\rho_d(n) \leqslant \rho_d^-(n)$ by Lemma 5.

5.2. $(d, 2d)$-Strings

For the computation of $\rho_d(2d)$ we can use overlapping r-covers satisfying additional conditions and hence refine the computation vis-à-vis the general case.

Definition 12. The r-cover $\{S_i = (s_i, e_i, p_i) \mid 1 \leqslant i \leqslant m\}$ of $x = x[1 \mathrel{..} n]$ satisfies the *parity condition* if for any $1 \leqslant i < m$, $\mathcal{A}(x[1 \mathrel{..} e_i - 1]) \cap \mathcal{A}(x[s_{i+1} + 1 \mathrel{..} n]) \subseteq \mathcal{A}(x[s_{i+1} \mathrel{..} e_i])$.

Lemma 13. *The singleton-free part of a run-maximal $(d, 2d)$-string x with all its singletons at the end has an r-cover satisfying the parity condition.*

Proof. Let x have $v \leqslant d - 2$ singletons, all at the end, and let $k(x)$ be the core vector of x. Assume that $k_i(x) = 0$ for some $1 \leqslant i \leqslant 2d - v$. If $i = 1$ or $2d - v$, then $\rho_d(2d) = \boldsymbol{r}(x) = \rho_d(2d-1) = \rho_{d-1}(2d-2)$, a contradiction, therefore $1 < i < 2d - v$. If $\mathcal{A}(x[1 \mathrel{..} i-1]) = \mathcal{A}(x[i+1 \mathrel{..} 2d-v])$, then $x[1 \mathrel{..} i-1]$ must have $d - v$ distinct symbols, and no singletons. Therefore, the length of both $x[1 \mathrel{..} i-1]$ and $x[i+1 \mathrel{..} 2d-v]$ must be at least $2(d-v)$, for a combined minimum length of $4(d-v)$. For these two substrings to fit within $x[1 \mathrel{..} 2d-v]$, we must have $4(d-v) \leqslant 2d - v$, a contradiction, as it implies there are more distinct characters in the string than there are singletons. Therefore, there is $\boldsymbol{c}$ so that either $\boldsymbol{c} \in \mathcal{A}(x[1 \mathrel{..} i-1]) \setminus \mathcal{A}(x[i+1 \mathrel{..} 2d-v])$ or $\boldsymbol{c} \in \mathcal{A}(x[i+1 \mathrel{..} 2d-v]) \setminus \mathcal{A}(x[1 \mathrel{..} i-1])$. Similarly as in the proof of Lemma 9, $\rho_d(2d) \leqslant \rho_d(2d-1) = \rho_{d-1}(2d-2)$, a contradiction. So every $k_i(x) \geqslant 1$ for $1 \leqslant i \leqslant 2d - v$, and thus $x[1 \mathrel{..} 2d-v]$ has an r-cover $\{S_i \mid 1 \leqslant i \leqslant m\}$ by Lemma 6. Assume that the r-cover does not satisfy the parity condition. There are two cases both yielding a contradiction:

(a) $\bigcup_{1 \leqslant i \leqslant t} S_i$ and $\bigcup_{t+1 \leqslant j \leqslant m} S_j$ for some $1 \leqslant t \leqslant m$ are disjoint and have at least one symbol $\boldsymbol{c}$ in common. If we replace all $\boldsymbol{c}$'s in $\bigcup_{1 \leqslant i \leqslant t} S_i$ by a new symbol $\hat{\boldsymbol{c}} \notin \mathcal{A}(x)$, we get a $(d+1, n)$-string y satisfying $\boldsymbol{r}(y) = \boldsymbol{r}(x)$. Thus $\rho_{d-1}(2d-2) = \rho_d(2d-1) = \rho_{d+1}(2d) \geqslant \boldsymbol{r}(y) = \boldsymbol{r}(x) = \rho_d(2d)$, a contradiction.

(b) $\bigcup_{1\leqslant i\leqslant t} S_i$ and $\bigcup_{t+1\leqslant j\leqslant m} S_j$ for some $1\leqslant t\leqslant m$ are overlapping, and there is a symbol $\boldsymbol{c}$ occurring in $\bigcup_{1\leqslant i\leqslant t} S_i$ and in $\bigcup_{t+1\leqslant j\leqslant m} S_j$, but not in the overlap of S_t and S_{t+1}. If we replace all $\boldsymbol{c}$'s in $\bigcup_{1\leqslant i\leqslant t} S_i$ by a new symbol $\hat{\boldsymbol{c}}\notin\mathcal{A}(x)$, we get a $(d+1,n)$-string y satisfying $\boldsymbol{r}(y)=\boldsymbol{r}(x)$. Thus $\rho_{d-1}(2d-2)=\rho_d(2d-1)=\rho_{d+1}(2d)\geqslant\boldsymbol{r}(y)=\boldsymbol{r}(x)=\rho_d(2d)$, a contradiction. □

Lemma 14. *If $\rho_{d'}(2d')=d'$ for any $d'<d$, then either $\rho_d(2d)=d$ or, for every run-maximal $(d,2d)$-string x with $v\leqslant d-2$ singletons all at the end, its singleton-free part $x[1\mathinner{\ldotp\ldotp}2d-v]$ has an overlapping r-cover satisfying the parity condition.*

Proof. The existence of the r-cover $\{S_i \mid 1\leqslant i\leqslant m\}$ of $x[1\mathinner{\ldotp\ldotp}2d-v]$ satisfying the parity condition follows from Lemma 13. We need to prove that either $\rho_d(2d)=d$ or there are no adjacent squares in the r-cover. Since $\rho_{d'}(2d')=d'$ for any $d'<d$, $\rho_{d'}(n')\leqslant n'-d'$ for any $n'-d'<d$. Assume that the r-cover of x has two adjacent squares S_t and S_{t+1}. Let $x_1=\bigcup_{1\leqslant i\leqslant t} S_i$ and let $x_2=\bigcup_{t<i\leqslant m} S_i$. Then $\boldsymbol{r}(x)=\boldsymbol{r}(x_1)+\boldsymbol{r}(x_2)$ and x_1 is a (d_1,n_1)-string for some d_1 and n_1, and x_2 is a (d_2,n_2)-string for some d_2 and n_2, where $n_1+n_2=2d-v$ and $d_1+d_2\geqslant d-v$. Since the r-cover satisfies the parity condition, $\mathcal{A}(x_1)$ and $\mathcal{A}(x_2)$ are disjoint and hence $d_1+d_2=d$. 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 $n_2-d_2>0$. As both n_1-d_1 and n_2-d_2 are smaller than d, $\rho_d(2d)=\boldsymbol{r}(x)=\boldsymbol{r}(x_1)+\boldsymbol{r}(x_2)\leqslant\rho_{d_1}(n_1)+\rho_{d_2}(n_2)\leqslant(n_1-d_1)+(n_2-d_2)=d$. □

Since the number of runs in a singleton-free $(d,2d)$-string is at most d, we do not need to consider the singleton-free strings. By Corollary 11, we can consider only $(d,2d)$-strings that have singletons at the end. Since $\rho_d(2d)>\rho_{d-1}(2d-2)$, we can set $\rho_d^-(2d)=\rho_{d-1}(2d-2)+1$ and thus consider only the strings that have the non-singleton part $\rho_d^-(2d)$-dense. By Lemma 14 we need only consider strings whose r-covers of the non-singleton part satisfy the parity condition with no adjacent squares. Moreover, we know that the number of singletons must be at least $\lceil\frac{7d}{8}\rceil$, see [3]. For every $\lceil\frac{7d}{8}\rceil\leqslant v\leqslant d-2$, let $\mathcal{T}_v$ denote the set of all singleton-free $\rho_d^-(2d))$-dense and r-covered $(d,2d-v)$-strings with overlapping r-cover satisfying the parity condition. Then

$$\rho_d(2d)=\max\left(d,\max\left\{\max_{x\in\mathcal{T}_v}\boldsymbol{r}(x)\colon\ \left\lceil\frac{7d}{8}\right\rceil\leqslant v\leqslant d-2\right\}\right).$$

6. Computational results

The described computational framework was implemented in C++, and was run in parallel on the SHARCNET computer cluster. We were able to recompute all previously known $\rho_2(n)$ values for $n\leqslant 60$ in a matter of hours, confirming the results reported by Kolpakov and Kucherov [7]. We were then able to extend the computations up to and including $n=74$. The new values are: $\rho_2(61)=52$, $\rho_2(62)=53$, $\rho_2(63)=54$, $\rho_2(64)=55$, $\rho_2(65)=\rho_2(66)=56$, $\rho_2(67)=57$, $\rho_2(68)=58$, $\rho_2(69)=59$, $\rho_2(70)=60$, $\rho_2(71)=61$, $\rho_2(72)=62$, $\rho_2(73)=63$, and $\rho_2(74)=64$. The results and sample run-maximal strings may be found at [1]. Whenever the computation required determining the number of runs in a concrete string, the C++ implementation of the Franek, Jiang, and Weng algorithm [6] was used. One particularly interesting string found is

aababaababbabaababaababbabaababaaaababaababbabaababaababbabaababaa,

which is a run-maximal $(2,66)$-string. This is, to the best of our knowledge, the first known example of a run-maximal string containing a run with exponent 4, in particular *aaaa*. It is also interesting to note that there are binary run-maximal strings which are themselves squares for $n=62$, 64, 66, 68, and 70.

7. Conclusion

We presented the notion of r-covers as a structural generalization of a uniform distribution of runs in a string. Then we showed that it is enough to consider overlapping r-covered strings in order to recursively determine the maximum number of runs $\rho_d(n)$. Based on these observations, we presented a computational framework with significantly reduced search space for computations of $\rho_d(n)$ based on the notion of density and exploiting the tightness of the available lower bound. As illustrations we obtained the previously unknown values of $\rho_2(n)$ for $61\leqslant n\leqslant 74$.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada and by the Canada Research Chair program, and made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (http://www.sharcnet.ca/). The authors would like to thank the referees for their comments that helped to improve the content and presentation of this paper.

References

[1] Andrew Baker, Antoine Deza, Frantisek Franek, Run-maximal strings, http://optlab.mcmaster.ca/~bakerar2/research/runmax/index.html.

[2] Andrew Baker, Antoine Deza, Frantisek Franek, On the structure of run-maximal strings, Journal of Discrete Algorithms 14 (2012) 10–14.

[3] Andrew Baker, Antoine Deza, Frantisek Franek, A parameterized formulation for the maximum number of runs problem, in: Jan Holub, Bruce Watson, Jan Žd'árek (Eds.), Festschrift for Bořivoj Melichar, Czech Technical University, Prague, Czech Republic, 2012, pp. 102–117.

[4] Maxime Crochemore, Lucian Ilie, Maximal repetitions in strings, Journal of Computer and System Sciences 74 (5) (2008) 796–807.

[5] Frantisek Franek, Jan Holub, A different proof of Crochemore–Ilie lemma concerning microruns, in: London Algorithmics 2008: Theory and Practice, College Publications, London, UK, 2009, pp. 1–9.

[6] Frantisek Franek, Mei Jiang, Chia-Chun Weng, An improved version of the runs algorithm based on Crochemore's partitioning algorithm, in: Proceedings of Prague Stringology Conference 2011, Prague, Czech Republic, 2011, pp. 98–105.

[7] Roman Kolpakov, Gregory Kucherov, personal communication.

[8] Roman Kolpakov, Gregory Kucherov, On maximal repetitions in words, in: Lecture Notes in Computer Science, vol. 1684, 1999, pp. 374–385.