WEAK REPETITIONS IN STRINGS

L. J. Cummings

Faculty of Mathematics
University of Waterloo

W. F. Smyth

Department of Computer Science & Systems
McMaster University

School of Computing
Curtin University of Technology

ABSTRACT

A weak repetition in a string consists of two or more adjacent substrings which are permutations of each other. We describe a straightforward $\Theta(n^2)$ algorithm which computes all the weak repetitions in a given string of length n defined on an arbitrary alphabet A. Using results on Fibonacci and other simple strings, we prove that this algorithm is asymptotically optimal over all known encodings of the output.

1 INTRODUCTION

Interest in the periodic behaviour of strings dates back to Thue [T06] at the turn of the century. Thue considered what we call here *strong repetitions* (equal adjacent substrings) and showed how to construct an infinitely long string on an alphabet of only three letters with no strong repetitions. (Other constructions on three letters have been discovered several times since, most recently by Dekking [D79] and Pleasants [P70] — the latter lists several references to earlier constructions.) More recently, Erdős [E61, p. 240] considered "Abelian squares" (what we call *weak repetitions*: adjacent substrings that are permutations of each other), and asked what was the minimum size of alphabet on which infinitely long strings with no weak repetitions could be constructed. In 1970 Pleasants [P70] gave a construction on an alphabet of five characters, and Keränen [K92] has very recently found a best possible construction on only four characters.

Typeset by $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX

It has been only in the last 15 years or so, with the increased modern emphasis on algorithms, that a problem more in the spirit of computer science has been considered: how to compute (efficiently) all the repetitions in a given string x of length n. It might be supposed that in the worst case such a computation would require time $\Omega(n^2)$, since it can easily be seen that the string $x = a^n$ contains $\lfloor n^2/4 \rfloor$ strong (also weak) repetitions. (For example, a^6 contains five distinct repetitions aa, three distinct repetitions a^2a^2, and one repetition a^3a^3.) However, in 1981 Crochemore [C81], using a clever *encoding* of repetitions (see the next section), devised a $\Theta(n \log n)$ algorithm to compute all the strong repetitions in a string x defined on an ordered alphabet. Crochemore also showed that, *in his encoding*, a Fibonacci string of length n contains $\Omega(n \log n)$ repetitions, so that, at least with respect to his encoding, his algorithm was "optimal". Somewhat later, two other, quite different, algorithms for computing all the strong repetitions were published [AP83, ML84], both also requiring $\Theta(n \log n)$ time, but now over an arbitrary alphabet.

This paper discusses, apparently for the first time, the computation of all the weak repetitions in x. This problem generalizes and includes the corresponding strong repetitions problem, since every strong repetition is also a weak one. In Section 2 we introduce some notation and terminology, in particular another encoding (called the $\mathcal{R}$-encoding) which appears to be more natural for weak repetitions. In Section 3 we then describe an algorithm for computing all the weak repetitions in x: this "obvious" algorithm executes in time $\Theta(n^2)$ on all strings of length n. In Section 4, the main part of the paper, we show that, in the $\mathcal{R}$-encoding, the Fibonacci string contains $\Theta(n^2)$ weak repetitions; further that, in Crochemore's encoding, another simple string contains $\Theta(n^2)$ weak repetitions. With respect to known encodings, therefore, we conclude that the computation of all weak repetitions is a $\Theta(n^2)$ problem. A final section gives some brief concluding remarks.

2 TERMINOLOGY & NOTATION

Let A denote a (possibly infinite) set of distinct elements a_i, $i = 1, 2, \ldots,$ which are not required to be ordered. We call A an *alphabet* and its elements *letters*. Let A^+ denote the set of all concatenations of elements of A, and let $A^* = \{\epsilon\} \cup A^+$, where ϵ denotes the empty element. The elements of A^* are called *strings*, and a string x of *length* $|x| = n \geq 1$ is written $x = x_1 x_2 \cdots x_n$, where each $x_i \in A$. If

$x = uv$, then u is said to be a *prefix* and v a *suffix* of x. For any positive integer k, a concatenation of k identical strings u is written u^k.

A string x is said to be *strongly periodic of order* k if there exists an integer $k > 1$ and a string $u \in A^+$ such that $x = u^k$. Similarly, x is said to be *weakly periodic of order* k if there exists $k > 1$ and $u_1 \in A^+$ such that $x = u_1 u_2 \cdots u_k$, where each u_i, $2 \leq i \leq k$, is a permutation of u_1 (that is, a concatenation of the same elements of A, but not necessarily in the same order). When $k = 2$ in these definitions, x is said to be a *strong* (respectively, *weak*) *square*. If x is not strongly (respectively, weakly) periodic of any order k, then we shall say that x is *strongly* (respectively, *weakly*) *primitive*. If there exists a strongly (respectively, weakly) periodic string w such that $x = uwv$ for some strings $u, v \in A^*$, then w is said to be a *strong* (respectively, *weak*) *repetition in* x. The following observations are immediate consequences of these definitions:

* if x is strongly periodic of order k, then x is weakly periodic of order k;
* if x is weakly primitive, then x is strongly primitive;
* if x is strongly (respectively, weakly) periodic of order k and $k' \mid k$, then x is strongly (respectively, weakly) periodic of order k';
* the number of weak repetitions in x is at least as great as the number of strong repetitions in x.

Consider some examples on the alphabet $A = \{a, b\}$: $x = abaababa$ is weakly primitive, therefore strongly primitive; $x = abbaabba$ is weakly periodic of order 4, hence weakly periodic of order 2, and is also strongly periodic of order 2; $x = bbaababa$ is strongly primitive and weakly periodic of order 2; $x = abbababababaab$ is weakly periodic of order 6, hence weakly periodic of orders 2 and 3. The string $x = abbaba$ contains strong (hence weak) repetitions b^2 and $(ba)^2$ and, in addition, the weak repetitions $(ab)(ba)$ and $(ab)(ba)(ba)$; as we have seen, the string $x = a^n$ contains exactly $\lfloor n^2/4 \rfloor$ strong (hence weak) squares.

Observing that it suffices to compute maximum-length repetitions of primitive substrings, Crochemore [C81] improves the definition of strong repetition as follows. Suppose there exist an integer $k > 1$, strings u and v, and a nonempty strongly primitive string z such that $x = uz^kv$ and z is neither a suffix of u nor a prefix of v. Then the strong repetition z^k is uniquely specified by the triple $(|u|+1, |z|, k)$, where $|u| + 1$ gives the *position* of the repetition, $|z|$ its *period*, and k its order. Clearly the collection of all such triples for a given string x specifies the strong repetitions of x; we call this collection the $\mathcal{C}$-*encoding* of the strong repetitions and denote it $\mathcal{C}(x)$. With the obvious adjustments, a $\mathcal{C}$-encoding of the weak repetitions can be defined in a similar way. Observe that for the string $x = a^n$, $\mathcal{C}(x) = \{(1,1,n)\}$ for both strong and weak repetitions: thus all the repetitions in x, including the $\lfloor n^2/4 \rfloor$ squares, are described by a single triple.

Other encoding schemes are possible for strong/weak repetitions. For instance, one may think of the c^{th} position of x as a *centre* of strong/weak squares of various lengths: then if a substring

$$x_{c,p}^- x_{c,p}^+ = (x_{c-p}x_{c-p+1}\cdots x_{c-1})(x_c x_{c+1}\cdots x_{c+p-1}) \tag{1}$$

were a strong/weak square of period $p \le p_c = \min\{c-1, n-c+1\}$ centred at c, it could be encoded by the pair (c,p). Clearly a collection of all such pairs (c,p) could be used to specify all the repetitions of x. This collection can be further compressed by taking advantage of cases where, for given c, the periods p fall into *ranges* of acceptable values; thus, for $p_2 \ge p_1$, the pairs $(c,p_1), (c,p_1+1), \ldots, (c,p_2)$ may be expressed as a range triple (c,p_1,p_2). A collection of such triples identifying all squares in x is called an $\mathcal{R}$-*encoding* of the repetitions and denoted $\mathcal{R}(x)$. For the string $x = a^n$, for example, a minimum-cardinality $\mathcal{R}$-encoding is given by $\mathcal{R}(x) = \{(c,1,p_c), c = 2,3,\ldots,n\}$, of cardinality $n-1$.

In [C81] it was shown that, for Fibonacci strings f_i, $i = 0,1,\ldots$, and for strong repetitions,

$$|\mathcal{C}(f_i)| \in \Omega(F_i \log F_i),$$

where F_i denotes $|f_i|$. (Fibonacci strings are defined on $A = \{a,b\}$ as follows: $f_0 = b$, $f_1 = a$; for every $i \ge 2$, $f_i = f_{i-1}f_{i-2}$.) It follows then that, with respect to the $\mathcal{C}$-encoding, the algorithms which compute strong repetitions in $O(n \log n)$ time are asymptotically optimal. In this paper we consider both the $\mathcal{C}$-encoding and the $\mathcal{R}$-encoding for weak repetitions, and exhibit classes of strings of length n such that both encodings necessarily contain $\Omega(n^2)$ elements; thus algorithms, such as the one described in Section 3, which compute weak repetitions in $O(n^2)$ time, are also, with respect to known encodings, asymptotically optimal.

3 A WEAK REPETITIONS ALGORITHM

Here we outline a simple $\Theta(n^2)$ algorithm (called Algorithm A) which computes a minimum-cardinality $\mathcal{R}$-encoding of the weak repetitions in a given string $x =$

$x_1x_2\cdots x_n$. We suppose that x contains exactly m distinct letters, which we denote by λ_i, $i = 1, 2, \ldots, m$. Clearly $m \le n$. The algorithm considers in turn each potential centre $c = 2, 3, \ldots, n$ of x to determine every integer $p \in [1, p_c]$ such that the pair (c, p) encodes a weak repetition. Recall that $p_c = \min\{c-1, n-c+1\}$.

Algorithm A makes use of two $O(n)$ integer arrays, $\Sigma[0..m]$ and INDEX$[1..n]$. For $i = 1, 2, \ldots, m$, $\Sigma[i]$ is used as a counter of the number of occurrences of λ_i: each occurrence to the left of c is counted with a decrement of 1, while each occurrence to the right is counted with an increment of 1. $\Sigma[0]$ is used as a "global" counter: as we shall see, $\Sigma[0] = 0$ if and only if a weak repetition has been found.

The array INDEX is used to specify positions in Σ, according to the following rule:

$$\text{INDEX}[j] = i \Longleftrightarrow x_j = \lambda_i.$$

Thus $\Sigma\big[\text{INDEX}[j]\big]$ is the counter corresponding to x_j, and so INDEX effectively replaces x, which is not mentioned at all in the main part of the algorithm.

The replacement of x by INDEX is performed in a preprocessing phase. Where x is defined on an arbitrary alphabet A, this replacement requires time $O(n^2)$; if A is totally ordered, the replacement can be effected (using a search tree, for example) in time $O(n \log n)$; if A is fixed and finite, conversion reduces to an $O(n)$ table lookup procedure.

Corresponding to each potential centre $c = 2, 3, \ldots, n$, Algorithm A computes a linked list L consisting of all ranges $[p_1, p_2]$ such that for every $p \in [p_1, p_2]$, (c, p) encodes a weak square. To accomplish this, the algorithm first initializes L to a single entry $[1, p_c]$ and then updates L by eliminating ranges which cannot give rise to weak repetitions. After all updates to L have been made, therefore, L consists of exactly those ranges of values of p which do give rise to weak repetitions. Over all possible values of c, the aggregate of these lists is equivalent to a minimum-cardinality $\mathcal{R}(x)$, and since L can contain at most $\lceil p_c/2 \rceil$ elements, it is clear that $|\mathcal{R}(x)| \in O(n^2)$. Moreover, since the algorithm handles the update of L without backtracking — that is, in monotone increasing order of p —, it follows that, for each c, update of L requires time $O(n)$ and, over all values of c, time $O(n^2)$.

Corresponding to each potential centre $c = 2, 3, \ldots, n$, Algorithm A first initializes all counters to zero, initializes L, and then, for each integer $p = 1, 2, \ldots, p_c$, decrements the counter $\Sigma\big[\text{INDEX}[c-p]\big]$ and increments $\Sigma\big[\text{INDEX}[c+p-1]\big]$. The entire processing for each centre c is as follows:

```
initialize all counters to zero; L ← [1, p_c];
for p ← 1 to p_c do
  i ← INDEX[c − p]; Σ[i] ← Σ[i] − 1;
  if Σ[i] ≥ 0 then
    Σ[0] ← Σ[0] − 1
  else
    Σ[0] ← Σ[0] + 1;
  i ← INDEX[c + p − 1]; Σ[i] ← Σ[i] + 1;
  if Σ[i] ≤ 0 then
    Σ[0] ← Σ[0] − 1
  else
    Σ[0] ← Σ[0] + 1;
  if Σ[0] ≠ 0 then
    delete p from L.
```

It is easy to see that $\Sigma[0] = 0$ after the processing for the current value of p if and only if (c, p) encodes a weak square. Over all values of c and p, the interior of the **for** loop for p will be executed once for each of exactly $\lfloor n^2/4 \rfloor$ position pairs $c-p$ and $c+p-1$; it follows that Algorithm A requires $\Theta(n^2)$ time. As we have seen, the additional space required for Algorithm A consists of L, Σ and INDEX, and is thus $\Theta(n)$.

As an example of the operation of this algorithm, suppose

$$x = f_6 = abaababaabaab$$

and consider $c = 7$ (so that $p_c = 6$). Then only for $p = 1$ and $p = 4$ does it occur that $\Sigma[0] \neq 0$: for $p = 1$, L becomes $\{(2,6)\}$ and for $p = 4$, L becomes $\{(2,3),(5,6)\}$. Thus the elements of $\mathcal{R}(x)$ which are output corresponding to $c = 7$ are $(7,2,3)$ and $(7,5,6)$.

The algorithm described here is an "obvious" algorithm, but it does not appear to be easy to improve on. We have devised two other algorithms as follows:

* Algorithm B, which, for each potential centre c, eliminates periods p from L which are inconsistent with the distribution of each individual letter λ_i in x;
* Algorithm C, which, for each c, eliminates from L all periods p which are inconsistent with a "balance" between *pairs* of letters found close to position c in x.

Neither of these algorithms can guarantee that backtracking will not occur in the update of L, and so each executes in time $O(mn^2)$. However, since it would not always be necessary to test all pairs of positions in x, it was expected that in many cases these algorithms would execute more quickly than Algorithm A. Timed runs of all the algorithms on long repetition-free and repetition-dense strings have not supported this expectation [TT93]: Algorithms B and C both appear to execute much more slowly on average than Algorithm A.

4 DISCUSSION OF COMPLEXITY

In this section we show that for Fibonacci strings f_n, $|\mathcal{R}(f_n)| \in \Omega(F_n^2)$, and also that for the strings $g_n = (aababbab)^n$ of length $8n$, $|\mathcal{C}(g_n)| \in \Omega(n^2)$. We conclude then that Algorithm A is asymptotically optimal over the $\mathcal{C}$- and $\mathcal{R}$-encodings of the output.

We consider first $g_n = (aababbab)^n$, a string of length $G_n = 8n$. In particular, we consider the weak repetitions of g_n as expressed in the $\mathcal{C}$-encoding; indeed, we initially confine our attention to those repetitions (i, p, k) where $i \equiv 1 \pmod 8$ and $p \equiv 1 \pmod 4$. We show first that for this special class of weak repetitions, it must be true that $k = 2$, and hence that there exist exactly $\binom{n+1}{2}$ of them.

Observe first that for $i \equiv 1 \pmod 8$, $g_n[i] = a$. Observe also that g_n may be written in the form

$$a(abab)(baba)(abab)(baba)\ldots(abab)bab,$$

so that for $p = 1, 5, 9, \ldots$, there exists a weak square (in fact a palindrome)

$$a(ababbaba)^{(p-1)/4}a, \tag{2}$$

provided that

$$i + 2p - 1 \leq 8n. \tag{3}$$

We see that each component of each square (2) necessarily contains $(p+1)/2$ a's and $(p-1)/2$ b's; that is, an excess of a's over b's of one. Furthermore, each such square is followed by substrings $b, ba, bab, babb\ldots$, each of which will contain at least as many b's as a's. Thus no squares (2) can be extended to cubes, from which we conclude that $k = 2$.

We wish now to count the number of occurrences ν_n of the weak squares $(i,p,2)$ in g_n. From (3) it follows that $p \le (8n-i+1)/2$, so that

$$\begin{aligned}\nu_n &= \sum_{i=1(8)}^{8n-7} \sum_{p=1(4)}^{(8n-i+1)/2} 1 \\ &= \sum_{i=1(8)}^{8n-7} (n-(i-1)/8) \\ &= n^2 - \sum_{i=1}^{n} (i-1) \\ &= \binom{n+1}{2}.\end{aligned}$$

Essentially the same argument, with the roles of a and b reversed, shows that for $i \equiv 5 \pmod 8$ and $p \equiv 1 \pmod 4$ there are another $\binom{n+1}{2}$ weak squares $(i,p,2)$. Similarly, the cases $i \equiv 3 \pmod 8$ and $i \equiv 7 \pmod 8$ with $p \equiv 3 \pmod 4$ add an additional $\binom{n+1}{2}$ and $\binom{n}{2}$ weak squares, respectively. Thus the total number of weak repetitions in the $\mathcal{C}$-encoding for odd positions i of g_n is $3\binom{n+1}{2} + \binom{n}{2} = 2n^2 + n$. We have then

Theorem 1. $|\mathcal{C}(g_n)| \in \Theta(G_n^2)$. $\square$

In fact, it is also true for the $\mathcal{R}$-encoding that $|\mathcal{R}(g_n)| \in \Theta(G_n^2)$. But it turns out in this case, due to the regularity of g_n, that a very slight modification of the $\mathcal{R}$-encoding can be used to reduce the output required to $\Theta(G_n)$. The modification required is to replace the triples (c, p_1, p_2) of the $\mathcal{R}$-encoding by quadruples (c, p_1, d, k) representing the squares

$$(c, p_1), (c, p_1 + d), \ldots, (c, p_1 + (k-1)d).$$

Therefore, in order to establish more clearly that the $\mathcal{R}$-encoding requires in the worst case output quadratic in the length of the string, we consider next the Fibonacci string f_n and show that $|\mathcal{R}(f_n)| \in \Omega(F_n^2)$.

The *Parikh* or *frequency vector* of a string $x = x_1 x_2 \cdots x_n$ over an alphabet A is an integer vector $\phi(x)$ of length $\alpha = |A|$, where the i^{th} element $\phi(x)[i]$ counts the number of occurrences in x of the i^{th} element of A. (For example, if $A = \{a, b\}$, then $\phi(a) = (1,0)$ and $\phi(b) = (0,1)$.) For strings x,y over A, it is easy to see that

$$\phi(xy) = \phi(x) + \phi(y).$$

Observe also that xy is a weak square if and only if $\phi(x) = \phi(y)$, so that in such a case $\phi(xy) = 2\phi(x)$. We state a special case of an important lemma on Sturmian strings which will be useful later:

Lemma 1. Let u and v denote any two substrings of equal length of a Fibonacci string. Then $\phi(u) - \phi(v)$ can only take one of the values $(0,0)$, $(-1,1)$, $(1,-1)$.

Proof. See [BS93]. $\square$

Let $ws(x)$ denote the number of weak squares $\big($of the form $(c,p)\big)$ in a string x. We now turn our attention to the estimation of $ws(f_n)$. Clearly $ws(f_n) \geq |\mathcal{R}(f_n)|$. In order to estimate more precisely, consider the two-dimensional array $T = T[1..F_n, 1..F_n]$ formed by applying the following rule:

$$\begin{aligned} T[c,p] &= 1, \text{ if } f_n \text{ contains a weak square } (c,p); \\ &= 0, \text{ otherwise.} \end{aligned}$$

Recall from Section 2 the definition of p_c, which for Fibonacci strings we modify to

$$p_c = \min\{c-1, F_n - c + 1\}.$$

Then clearly for every $p > p_c$, $T[c,p] = 0$, so that row c of T contains at most p_c nonzero entries and column p is all zeros for every $p > F_n/2$. Observe also that for integers p such that $1 \leq p \leq F_n/2$, column p of T contains at most $F_n - 2p + 1$ nonzero entries. Since the number of weak squares is just the number of ones in T, we can then easily compute a crude upper bound for $ws(f_n)$:

Lemma 2. $ws(f_n) \leq \lfloor F_n^2/4 \rfloor$.

Proof. The upper bound is just the sum of the possibly nonzero entries in the columns of T. When F_n is even, this sum is

$$(F_n - 1) + (F_n - 3) + \cdots + 1 = F_n^2/4;$$

and when F_n is odd, the sum is

$$(F_n - 1) + (F_n - 3) + \cdots + 2 = (F_n^2 - 1)/4.$$

Both these sums reduce to $\lfloor F_n^2/4 \rfloor$. $\square$

Obviously, the upper bound in Lemma 2 is far from being best possible. For example, f_8 has length $F_8 = 34$ and contains 136 weak squares, but the bound provided by Lemma 2 is 289. In order to compute more precise bounds on $ws(f_n)$, we consider now what may be called the *diagonals* of the array T. These are ordered collections of the values of all those positions in T which may possibly take the value 1; they are defined as follows:

$$D_c : \{T[c, c-1], T[c+1, c-2], \ldots, T[2c-2, 1]\}, \tag{4}$$

where $c = 2, 3, \ldots, M$, with $M = \lceil F_n/2 \rceil$ if F_n is odd and $\lceil F_n/2 \rceil + 1$ otherwise; and

$$D'_c : \{T[c+1, c-1], T[c+2, c-2], \ldots, T[2c-1, 1]\}, \tag{5}$$

for $c = 2, 3, \ldots, \lceil F_n/2 \rceil$. The collections D_c and D'_c are interleaved cross-diagonal entries that together fill a triangle of T whose sides are the first column, the main cross-diagonal, and the first diagonal below the main diagonal. Observe that $|D_c| = |D'_c| = c-1$. From now on we shall treat the D_c and D'_c simply as strings of length $c-1$ defined on the alphabet $A = \{0, 1\}$.

The following lemma shows that adjacent positions in any D_c or D'_c can be both zero or both one only if the substring aa occurs at a specified location in f_n. This will pave the way for showing that approximately half of the entries in each D_c or D'_c are ones, hence that the number of weak squares in f_n is order F_n^2.

Lemma 3. Suppose x is any Fibonacci string. For integers $c \in [3, F_n - 1]$ and $p \in [2, p_c]$, let $h_1 = T[c, p]$ and $h_2 = T[c+1, p-1]$ denote adjacent positions in some diagonal D_c or D'_c of the array T. Then $h_1 = h_2$ if and only if $x_{c-p} = x_{c-p+1} = a$.

Proof. Let q denote the Parikh vector of x_c and let q' denote the Parikh vector of $x_{c-p}x_{c-p+1}$. Observe that $q = (0,1)$ or $(1,0)$ and that, since b^2 never occurs in any Fibonacci string, $q' \neq (0,2)$. It follows that $q' = (2,0)$ if and only if $x_{c-p} = x_{c-p+1} = a$. Recall the notation $x^-_{c,p}$ and $x^+_{c,p}$ introduced in (1). Now let

$$\delta_1 = \phi(x^-_{c,p}) - \phi(x^+_{c,p}), \tag{6}$$

$$\delta_2 = \phi(x^-_{c+1,p-1}) - \phi(x^+_{c+1,p-1}), \tag{7}$$

and observe by Lemma 1 that δ_1 and δ_2 can assume only the values $(0,0)$, $(1,-1)$, or $(-1,1)$. From (6) it follows that

$$\phi(x^+_{c+1,p-1}) = \phi(x^+_{c,p}) - q,$$

$$\phi(x^-_{c+1,p-1}) = \phi(x^+_{c,p}) + \delta_1 + q - q',$$

and so (7) implies that

$$q' = (\delta_1 - \delta_2) + 2q. \tag{8}$$

First consider the case $h_1 = h_2 = 1$; that is, $\delta_1 = \delta_2 = (0,0)$. Then (8) implies that $q' = 2q$ and, since $q' \neq (0,2)$, it follows that $q' = (2,0)$.

Next suppose that $h_1 = h_2 = 0$, so that neither δ_1 nor δ_2 can equal $(0,0)$. Then if $\delta_1 = \delta_2$, (8) tells us again that $q' = (2,0)$; while otherwise $\delta_1 = -\delta_2$, so that (8) reduces to $q' = 2(\delta_1 + q)$, once more implying that $q' = (2,0)$.

Conversely, when $h_1 = 1$ and $h_2 = 0$, it follows from (8) that $q' = 2q - \delta_2$, where $\delta_2 = (-1,1)$ or $(1,-1)$; this equality can hold only if $q' = (1,1)$. We reach the same conclusion in the case $h_1 = 0$, $h_2 = 1$. Since all possibilities have been considered, the result is proved. $\square$

Lemma 4. Suppose x is any Fibonacci string. Let d denote any bit string (4) or (5) of length $c-1$ corresponding to x, and suppose that $\phi(d) = (i,j)$, where i counts the frequency of zeros and j the frequency of ones. Then

(a) $j = i+1$ if and only if c is even and

$$x^-_{c+1,c-1}x^+_{c+1,c-1} = x_1x_2\cdots x_{2c-2}$$

has suffix aa;

(b) $i = j+2$ if and only if c is odd and $x^-_{c+1,c-1}x^+_{c+1,c-1}$ has suffix $aaba$;

(c) $j \leq i \leq j+1$, otherwise.

Proof. Suppose first that $d = D_c$ for some valid integer c. To exclude trivial cases, suppose that $c \geq 3$. Then the $c-1$ entries d_h, $h = 1,2,\ldots,c-1$, in d are 1 or 0 according as the $c-1$ substrings

$$x^-_{c,c-1}x^+_{c,c-1} = (x_1x_2\cdots x_{c-1})(x_c\cdots x_{2c-2})$$

$$x^-_{c+1,c-2}x^+_{c+1,c-2} = (x_3x_4\cdots x_c)(x_{c+1}\cdots x_{2c-2})$$

$$\vdots$$

$$x^-_{2c-2,1}x^+_{2c-2,1} = x_{2c-3}x_{2c-2}$$

are squares or not, respectively. Observe that $d_h = T[c+h-1, c-h]$. Therefore, by Lemma 3, consecutive entries d_h and d_{h+1}, $1 \leq h \leq c-2$, are unequal if and only if $x_{2h-1} \neq x_{2h}$. Thus consecutive entries in d flipflop (from 0 to 1, or from 1 to 0) as determined by the first $c-2$ pairs of entries in x:

$$x_1x_2, x_3x_4, \ldots, x_{2c-5}x_{2c-4}.$$

Consider the case in which one of these pairs is aa. Occurrences of aa cannot exist either at the beginning or at the end of x, and in fact must always be embedded in substrings $x' = abaaba$; that is, preceded by a pair ab and followed by a pair ba. Thus, except in the case that the substring x' in question is a terminating substring (suffix) of $x^-_{c,c-1}x^+_{c,c-1} = x_1x_2\cdots x_{2c-2}$, the entries in d corresponding to x' must be either 1001 or 0110, depending on whether or not the substring marks the beginning of a square in x. In each of these cases, the number of zeros equals the number of ones, and the final entry equals the initial one. Since pairs in x which are not aa must be either ab or ba, each of which causes a flipflop in d, it follows that, except when x' is a suffix, the number of ones and the number of zeros in d can differ by at most one. In particular, for any even position $h < c-1$,

$$\phi(d_1d_2\cdots d_h) = (h/2, h/2), \tag{9}$$

a fact used below.

Now consider the case in which x' is a suffix of $x_1x_2\cdots x_{2c-2}$. In this case the final entries in d are either 100 or 011. But observe in particular that the final entry d_{c-1} in d is determined by whether or not $x_{2c-3} = x_{2c-2}$; that is, whether or not $b = a$. Thus $d_{c-1} = 0$, so that only the case 100 is possible. In this case, if in addition c is odd, it follows from (9) that $\phi(d_1d_2\cdots d_{c-3}) = (\frac{c-3}{2}, \frac{c-3}{2})$, and so $i = (c+1)/2 = j+2$. (That this case actually arises may be seen by considering f_8 and $c = 13$.)

Finally, consider the only remaining case: aa a suffix of $x_1x_2\cdots x_{2c-2}$. This is the only case in which $d_{c-1} = 1$, and, since aa is always preceded by ab, it follows that $d_{c-2} = 0$. Thus when c is odd, $\phi(d) = (\frac{c-1}{2}, \frac{c-1}{2})$, while when c is even, $\phi(d) = (\frac{c}{2}-1, \frac{c}{2})$, so that $j = i+1$.

Thus the result is proved for $d = D_c$. An almost identical argument establishes the result also for $d = D'_c$. $\square$

We remark now that in the strings D_c and D'_c, every instance in which case (a) of Lemma 4 holds is matched by an instance of case (b), and vice versa. That is, c is odd and $x_1x_2\cdots x_{2c-2}$ has suffix $aaba$ if and only if $c-1$ is even and $x_1x_2\cdots x_{2c-4}$

has suffix aa. It follows that in counting the cumulative frequency of ones in the D_c and in the D'_c, we can simply ignore cases (a) and (b), and count $\lfloor (c-1)/2 \rfloor$ ones in each of these strings. The total number of ones in T is then just the sum of $\lfloor (c-1)/2 \rfloor$ over all strings D_c and D'_c, where c takes the values specified in (4) and (5). For example, for $F_n \equiv 3 \pmod 4$, it is not difficult to compute that

$$\begin{aligned} ws(f_n) &= \sum_{k=1}^{(F_n-3)/4} k \\ &= (F_n^2 - 2F_n - 3)/8. \end{aligned}$$

Similar calculations may be carried out for $F_n \equiv 0, 1, 2 \pmod 4$, yielding

Theorem 2. $ws(f_n) = (F_n^2 - 2F_n + q)/8$, where
 (a) $q = 0$ if $F_n \equiv 0 \pmod 2$;
 (b) $q = 1$ if $F_n \equiv 1 \pmod 4$;
 (c) $q = -3$ if $F_n \equiv 3 \pmod 4$. $\square$

This result specifies the number of weak squares (c, p) in f_n. However, the question remains whether, by encoding every collection $(c, p_1), (c, p_1+1), \ldots, (c, p_2)$ of weak squares as a single triple (c, p_1, p_2), an algorithm could perhaps run faster than $O(F_n^2)$; that is, in time proportional to something less than the square of the string length. For example, in f_6 the weak repetitions can be encoded by only 10 triples: $(4,1,3)$, $(6,2,3)$, $(6,5,5)$, $(7,2,3)$, $(7,5,6)$, $(9,1,3)$, $(9,5,5)$, $(10,3,3)$, $(11,3,3)$, $(12,1,2)$. Without the use of the triples, 18 pairs (c, p) would be required.

Observe that any one of these output triples, say (c, p_1, p_2), corresponds to a sequence, or *run*, of one or more consecutive ones in row c of the matrix T; specifically,

$$T[c, p_1] = T[c, p_1 + 1] = \cdots = T[c, p_2] = 1,$$

where $T[c, p_2 + 1] = 0$ and also

$$p_1 > 1 \implies T[c, p_1 - 1] = 0.$$

Thus whenever 01 occurs in row c of T, a run (triple) is beginning, and whenever 10 occurs, a run (triple) is ending. Therefore, to determine a lower bound on the number of output triples, we may count the occurrences of 01 or of 10. As it turns out, it is convenient (and sufficient) to count the total occurrences of both 01 and 10, and then divide by two; the following technical lemma provides the basis for doing this.

Lemma 5. Let x denote any Fibonacci string, and let $i \geq 1$ and $j \geq i+3$ denote any two nonadjacent positions in x such that $j-i$ is odd. Let

$$c = (i+j+1)/2,\ p = (j-i-1)/2.$$

Then $T[c,p] = T[c,p+1]$ if and only if $x_i = x_j$.

Proof. Since the occurrences are nonadjacent, and since $j-i$ is odd, it follows that a substring w of even length lies between positions i and j. In fact, $w = x^-_{c,p}x^+_{c,p}$, where c and p are as defined in the statement of the lemma.

Suppose first that $x_i = x_j$, and consider the case in which w is a weak square. Then $T[c,p] = 1$. But since $x_i = x_j$, it follows that $T[c,p+1] = 1$ also. Similarly, when w is not a weak square, it is clear that $T[c,p] = T[c,p+1] = 0$.

Conversely, suppose that $T[c,p] = T[c,p+1]$. If w is a weak square, then we see that x_iwx_j must be also, and so we conclude that $x_i = x_j$. Similarly, if w is not a weak square, then neither is x_iwx_j, and so it follows from Lemma 1 that, in this case also, $x_i = x_j$. $\Box$

Lemma 5 tells us that by counting all the pairs (i,j), $j-i \geq 3$, for which $x_i \neq x_j$ and $j-i$ is odd, we will identify all cases in which $T[c,p] \neq T[c,p+1]$; that is, all occurrences of 01 (beginning of a run of ones) and of 10 (end of a run of ones) in the triangle of T determined by the strings (4) and (5). These occurrences do not include all beginnings and all ends of runs; specifically excluded are beginnings of runs for which $p = 1$ (corresponding to occurrences of aa in x) and endings of runs for which $p = p_c$. Thus the number of pairs (i,j) is only a lower bound on the number of runs; nevertheless, as we shall now show, this number is $\Theta(|x|^2)$.

Suppose that some Fibonacci string f_n is given, $n \geq 3$. It is easy to show that b occurs F_{n-2} times in f_n, and so it follows that there are F_{n-1} occurrences of a. Let $m > 1$ denote the number of b's at odd positions of f_n; then $F_{n-2} - m$ b's occur at even positions. Note that there are $\lceil F_n/2 \rceil$ odd positions and $\lfloor F_n/2 \rfloor$ even positions in f_n. Hence there are $\lceil F_n/2 \rceil - m$ a's at odd positions and $\lfloor F_n/2 \rfloor - F_{n-2} + m$ a's at even positions.

To simplify the computation a little, let us assume that n is odd, so that f_n ends in a and every occurrence of b has exactly two neighbouring occurrences of a. It is these two neighbouring occurrences that are excluded by the "nonadjacent" condition of Lemma 5. Then over all b's occurring at odd positions, the total number of nonadjacent pairs with a's occurring at even positions is given by

$$m\big(\lfloor F_n/2\rfloor - F_{n-2} + m - 2\big).$$

Similarly, the total number of nonadjacent pairs corresponding to b's at even positions and a's at odd positions is

$$\big(F_{n-2} - m\big)\big(\lceil F_n/2\rceil - m - 2\big).$$

Then $|\mathcal{R}(f_n)|$, the total number of runs of ones in T, is at least

$$\begin{aligned}
&\frac{1}{2}\Big\{m\big(\lfloor F_n/2\rfloor - \lceil F_n/2\rceil - F_{n-2} + 2m\big) + F_{n-2}\big(\lceil F_n/2\rceil - m - 2\big)\Big\}\\
&> \frac{1}{2}\{F_{n-2}(F_n/2 - 2) - m(2F_{n-2} - 2m + 2)\}\\
&= m^2 - (F_{n-2} + 1)m + F_{n-2}(F_n/2 - 2)/2\\
&\equiv g(m).
\end{aligned}$$

The function $g(m)$ achieves its minimum value if

$$\frac{dg(m)}{dm} = 2m - (F_{n-2} + 1) = 0;$$

that is, if $m = (F_{n-2} + 1)/2$. In this case,

$$g(m) = \big(F_{n-2}(F_{n-1} - 6) - 1\big)/4.$$

Since $F_{n-2} > F_n/3$, it follows that for sufficiently large n, $g(m) > F_n^2/36$, and hence that $|\mathcal{R}(f_n)| \in \Omega(F_n^2)$. Since $|\mathcal{R}(f_n)| < ws(f_n)$, so that by Theorem 2 $|\mathcal{R}(f_n)| \in O(F_n^2)$, we have thus proved

Theorem 3. $|\mathcal{R}(f_n)| \in \Theta(F_n^2)$. $\square$

In fact, it appears that, making use of more precise calculations, it is possible to establish that $|\mathcal{R}(f_n)| \approx ws(f_n)/2$.

5 CONCLUDING REMARKS

In this paper we have presented a simple $\Theta(n^2)$ algorithm for finding all the weak repetitions in a given string x of length n. We have shown that this algorithm is optimal over known encodings of the output; in the course of doing so, we have derived an exact expression for the number of weak squares in a Fibonacci string.

We remark that the methodology used to count weak squares and weak repetitions in Fibonacci strings may also have applications to similar counting problems on other strings.

The results of this paper suggest, but do not clearly establish, that the computation of the weak repetitions in x is an $\Omega(n^2)$ problem. To prove this conjecture, it would be necessary to find an information-theoretic argument that would show that $\Theta(n^2)$ processing steps are required in the worst case. In fact, an even stronger result has been proved for the strong repetitions problem [ML84]: Main and Lorentz show that, over a possibly infinite alphabet, $\Omega(n \log n)$ time is required to determine whether or not x contains a strong repetition. We give here a somewhat different proof which applies also to weak repetitions.

For a string x of length n, suppose that $n = 2^k$ for some positive integer k, and suppose further that the letter $x_{n/2}$ does not appear in $x_{n/2+1}x_{n/2+2}\cdots x_n$. Suppose in fact that this property applies recursively to substrings of x of length $2^{k-1}, 2^{k-2}, \ldots, 1$. It follows then that any weak (or strong) repetition in x occurs either in the substring $x_1x_2\cdots x_{n/2}$ or in $x_{n/2+1}x_{n/2+2}\cdots x_n$. In order to verify this fact, it is both necessary and sufficient to perform $n/2$ comparisons of $x_{n/2}$ against $x_{n/2+1}, x_{n/2+2}, \ldots, x_n$. Let $c(n)$ denote the number of comparisons required to perform the verification. Then

$$c(n) = 2c(n/2) + n/2,$$

a recurrence relation which can easily be solved, using the initial condition $c(1) = 0$, to yield

$$c(n) = \frac{n}{2}\log_2 n.$$

Hence

Theorem 4. Let x be a string of length n. The time required to determine whether or not x contains a (strong or weak) repetition is $\Omega(n \log n)$. $\Box$

It appears likely that for weak repetitions the lower bound of Theorem 4 can be increased to $\Omega(n^2)$. If so, then it would follow that any other weak repetitions algorithms would necessarily require $\Theta(n^2)$ time.

REFERENCES

[**AP83**] A. Apostolico & F. P. Preparata, **Optimal off-line detection of repetitions in a string**, *Theoretical Comp. Sci.* 22 (1983) 297-315.

[**BS93**] J. Berstel & P. Séé, **A characterization of Sturmian morphisms**, *The Mathematical Foundations of Computer Science*, A. Borzyszkowski & S. Sokolowski (eds.), Springer-Verlag (1993) 281-290.

[**C81**] M. Crochemore, **An optimal algorithm for computing the repetitions in a word**, *Inf. Proc. Lett.* 12-5 (1981) 244-250.

[**D79**] F. M. Dekking, **Strongly non-repetitive sequences and progression-free sets**, *JCT Series "A"* 27 (1979) 181-185.

[**E61**] P. Erdős, **Some unsolved problems**, *Hungarian Academy of Sciences Mat. Kutató Intézet Kőzl.* 6 (1961) 221-254.

[**K92**] V. Keränen, **Abelian squares are avoidable on 4 letters**, *Lecture Notes in Computer Science* 623, Springer-Verlag (1992) 41-52.

[**ML84**] M. G. Main & R. J. Lorentz, **An $O(n\mathbf{log}n)$ algorithm for finding all repetitions in a string**, *J. Algs.* 5 (1984) 422-432.

[**P70**] P. A. Pleasants, **Non-repetitive sequences**, *Proc. Cambridge Phil. Soc.* 68 (1970), 267-274.

[**T06**] A. Thue, **Über unendliche Zeichenreichen**, *Norske Vid. Selsk. Skr. I, Mat. Nat. Kl. Christiana* 7 (1906) 1-22.

[**TT93**] C. Y. Tan & T. K. K. Teo, honours project, School of Computing, Curtin University (1993).

ACKNOWLEDGEMENTS

The work of both authors was supported in part by grants from the Natural Sciences & Engineering Research Council of Canada. The authors express their appreciation of the work of Chin Yong Tan and Kelvin Teo of Curtin University, who programmed and tested the algorithms discussed in this paper.

Revised 8 November 1994