Algorithms for Finding a Minimum Repetition Representation of a String

Atsuyoshi Nakamura[1], Tomoya Saito[1], Ichigaku Takigawa[2], Hiroshi Mamitsuka[2], and Mineichi Kudo[1]

[1] Hokkaido University, Kita 14, Nishi 9, Kita-ku, Sapporo 060-0814, Japan, {atsu@main.,saito@,mine@main.}ist.hokudai.ac.jp,
[2] Institute for Chemical Research, Kyoto University, Uji, Kyoto 611-0011, Japan, {mami,takigawa}@kuicr.kyoto-u.ac.jp

Abstract. A string with many repetitions can be written compactly by replacing h-fold contiguous repetitions of substring r with $(r)^h$. We refer to such a compact representation as a *repetition representation string* or RRS, by which a set of disjoint or nested tandem arrays can be compacted. In this paper, we study the problem of finding a *minimum RRS* or MRRS, where the size of an RRS is defined to be the sum of its component letter sizes and the sizes needed to describe the repetitions $(\cdot)^h$ which are defined as $w_R(h)$ using a repetition weight function w_R. We develop two dynamic programming algorithms to solve the problem. One is CMR that works for any repetition weight function, and the other is CMR-C that is faster but can be applied only when the repetition weight function is constant. CMR-C is an $O(w(n+z))$-time algorithm using $O(n+z)$ space for a given string with length n, where w and z are the number of distinct primitive tandem repeats and the number of their occurrences, respectively. Since $w = O(n)$ and $z = O(n \log n)$ in the worst case, CMR-C is an $O(n^2 \log n)$-time $O(n \log n)$-space algorithm, which is faster than CMR by $((\log n)/n)$-factor.

Keywords: tandem repeat, string algorithm

1 Introduction

A contiguous repeat of a substring embedded in a string is called a *tandem array*, or a *tandem repeat* (or a *square*) for a two-fold repetition. The problem of finding tandem arrays and repeats has been studied for more than two decades in the fields of computer science, mathematics and biology [4]. Efficient algorithms for finding *all* or *primitive* tandem repeats, namely, repeats whose repeated units themselves are not tandem arrays, have been developed so far [2–4].

In this paper, we consider *repetition representation strings*, or RRSs, which allows to use notations $(r)^h$ instead of contiguous sequences $rr \cdots r$ of h identical strings r. An RRS can represent many tandem arrays simultaneously but cannot always represent all tandem arrays in a string. An RRS is a string representation for a set of *disjoint* or *nested* tandem arrays in a string, where two substrings of

a string are disjoint if they have no intersection, and $(r)^h$ and $(r')^{h'}$ are nested if r contains $(r')^{h'}$. Different RRSs are possible for a string depending on which set of tandem arrays is represented. As an evaluation measure for RRSs, we use their sizes, that is, we consider the problem of finding a *minimum RRS* (MRRS) for a given string. Our expectation that an MRRS represents an essential repetition structure of a string is supported by the well-known MDL principle: "the success in finding regularities can be measured by the length with which the data can be described" (www.mdl-research.org).

We define the size of an RRS as the sum of its component sizes, namely, the sum of its component letter sizes and the sizes needed to describe the repetitions $(\cdot)^h$ which are defined as $w_R(h)$ using a repetition weight function w_R.

We developed two algorithms: CMR and CMR-C. CMR works for any repetition weight function w_R, though it is slow; it runs in $O(n^3)$ time and $O(n^2)$ space. On the other hand, CMR-C works only when the repetition weight function is constant, but it is faster; it is an $O(w(n+z))$-time $O(n+z)$-space algorithm using a sophisticated technique of finding tandem repeats. Here, n is the length of a given string, z is the number of (occurrences of) primitive tandem repeats in the string, and w is the number of distinct strings among them. Considering the worst case setting [1, 2]: $w = O(n)$ and $z = O(n \log n)$, these complexities correspond to $O(n^2 \log n)$ time and $O(n \log n)$ space, which are smaller than those of CMR by $((\log n)/n)$-factor. We further note that CMR-C is more effective for smaller w and z.

2 Problem Setting

Let Σ be a finite *alphabet*, whose elements are called *letters*. A *string* s is a sequence of letters with finite length. The length of string s is the number of letters in s which is denoted by $|s|$. More general notion called a *repetition representation string*, or an RRS, is defined as follows. First, all strings are RRSs themselves. If r_1 and r_2 are RRSs, then $r_1 r_2$, concatenation of r_1 and r_2, is also an RRS. If r is an RRS, then $(r)^h$ $(h \geq 2)$ is also an RRS that is another representation of a concatenation $rr \cdots r$ of h identical RRSs r. Note that parentheses '(', ')' and codes for numbers are special symbols not contained in Σ. A substring represented by RRS $(r)^h$ is called a *tandem array*, and it is called a *tandem repeat* when $h = 2$. We say that $(r)^h$ is *expanded* to the concatenation $rr \cdots r$ of h identical RRSs r and reversely $rr \cdots r$ is *reduced* to $(r)^h$. String s without $(\cdot)^h$ notation is said to be *represented by an RRS* r if s is obtained by expanding all the $(\cdot)^h$ in r for any h.

The *size* of an RRS can be calculated using given two non-negative weight functions, *alphabet weight function* w_Σ on Σ and *repetition weight function* w_R on the set of natural numbers at least 2. Given w_Σ and w_R, the size $l(r)$ of an RRS r is recursively defined as follows:

$$
\begin{aligned}
l(\lambda) = 0 \text{ for empty string } \lambda,\ \ l(a) = w_\Sigma(a) \text{ for } a \in \Sigma,\\
l(r_1 r_2) = l(r_1) + l(r_2) \text{ for all RRSs } r_1 \text{ and } r_2, \text{ and}\\
l((r)^h) = l(r) + w_R(h).
\end{aligned}
$$

Remark 1. In most cases, we use constant functions as w_Σ and w_R for simplicity. A constant function w_R in the uniform cost model and $w_R(h) = \log h + c$ in the logarithmic cost model seem natural, where c is a some constant. In practice, w_Σ and w_R should be decided depending on applications.

The problem we deal with in this paper is the following one.

Problem 1. Given a string s, an alphabet weight function w_Σ and a repetition weight function w_R, find a minimum(-sized) RRS r that represents s.

3 Algorithms

In this section, we show algorithms for problem 1. We assume that s is an arbitrary string $a_1 a_2 \cdots a_n$ with length n. For $1 \leq i \leq j \leq n$, we let $s[i..j]$ denote the substring $a_i a_{i+1} \cdots a_j$ of s, and let $r_*[i..j]$ denote a minimum RRS (MRRS) representing $s[i..j]$ in the following subsections.

3.1 General Algorithm

First, we show algorithm CMR that works for any repetition weight function w_R. CMR calculates an MRRS representing a given string $s[1..n]$ by calculating MRRSs representing $s[i..j]$ for all the substrings $s[i..j]$ of $s[1..n]$ using dynamic programming. Its dynamic programming is based on the following proposition, which says that an MRRS representing a string is either a repetition of an MRRS representing its some prefix or a concatenation of two MRRSs representing its two substrings that are made by cutting the string at some position.

Proposition 1. *For all $1 \leq i < j \leq n$, $r_*[i..j]$ is one of the followings:*

(1) $(r_*[i..i+d-1])^h$ for some $d \geq 1$ and $h \geq 2$ with $hd = j - i + 1$,
(2) $r_*[i..i+d]r_*[i+d+1, j]$ for some $0 \leq d < j - i$.

For given $1 \leq i < j \leq n$, assume that $r_*[i'..j']$ is already known for all $1 \leq i' < j' \leq n$ with $j' - i' < j - i$. By the proposition, $r_*[i..j]$ is obtainable by searching an MRRS among at most $\lfloor (j - i + 1)/2 \rfloor$ possibilities of (1) and $j - i$ possibilities of (2). If $l(r_*[i'..j'])$ for all $1 \leq i' < j' \leq n$ with $j' - i' < j - i$ is already calculated, the size for each possible RRS can be calculated in constant time. Thus, such search can be finished in $O(j - i)$ time. This means that an MRRS $r_*[1..n]$ for a given string $s[1..n]$ is obtainable in $O(n^3)$ time and $O(n^2)$ space by dynamic programming using the fact that $r_*[i..i] = s[i..i] = a_i$ for all $1 \leq i \leq n$.

3.2 Algorithm for a Constant Repetition Weight Function

In this subsection, we describe an algorithm faster than CMR in the case with a constant repetition weight function. In this subsection, w_R is always supposed to be a constant function.

First, note that

$$l((\underbrace{rr\cdots r}_{h \text{ times}})^g) = hl(r) + w_R(g) \geq l(r) + w_R(hg) = l((r)^{hg})$$

because w_R is supposed to be a constant function. This means that we only have to check combinations of *primitive* tandem arrays, where a tandem array $(r)^h$ is said to be primitive if r cannot be represented by RRS $(r')^{h'}$ for any string r' and $h' \geq 2$. Such a tandem array is called a *primitive* tandem repeat when $h = 2$. It is known that there are at most $O(n \log n)$ *occurrences* of primitive tandem repeats in a string of length n.

Let $E(i)$ denote the set of primitive tandem repeats ending at the ith letter of s. For each tandem repeat $s[i-2j+1..i]$ in $E(i)$, namely, for each tandem repeat with width j and ending at i, $h_*(i,j)$ denotes an optimal number of repetitions of the last j letters as an RRS-representation of $s[1..i]$:

$$h_*(i,j) \stackrel{\text{def}}{=} \arg\min_g \{l(r(g)) : \mu(r(g)) = s[1..i], \\ r(g) = r_*[1..i-gj](r_*[i-j+1..i])^g\},$$

where $\mu(r)$ denotes an expanded string of r. Then, $r_*[1..i-h_*(i,j)j](r_*[i-j+1..i])^{h_*(i,j)}$ is a minimum RRS representing $s[1..i]$ among the RRSs of the form $r_*[1..i-gj](r_*[i-j+1..i])^g$ $(g \geq 2)$ by the definition, and also a minimum RRS even among all the RRSs of the form $r[1..i-gj](r[i-j+1..i])^g$ $(g \geq 2)$ for any RRSs $r[1..i-gj]$ and $r[i-j+1..i]$ representing $s[1..i-gj]$ and $s[i-j+1..i]$, respectively. Fortunately, $h_*(i,j)$ can be calculated efficiently using dynamic programming.

The following proposition indicates that candidate RRSs for $r_*[1..i]$ can be narrowed to only $|E(i)|+1$ RRSs.

Proposition 2. *One of the RRSs of the following form (1) or (2) is an MRRS representing* $s[1..i]$.

(1) $r_*[1..i-1]a_i$,
(2) $r_*[1..i-h_*(i,j)j](r_*[i-j+1..i])^{h_*(i,j)}$ *for* $s[i-2j+1..i] \in E(i)$.

By the above propositions, $r_*[1..n]$ can be calculated using dynamic programming if an appropriate representation of $E(i)$ is prepared. We call such a dynamic programming algorithm FindBestComb. As an input of FindBestComb, $E(i)$ is assumed to be given by a linked list, where the list entry for tandem repeat $s[i-2j+1..i]$ has the repeat width j, an MRRS $r_*[i-j+1..i]$ and its size, and the pointer to the entry for the tandem repeat $s[i-3j+1..i-j]$. FindBestComb calculates $l(r_*[1..i])$ in the increasing order of i. For each i, it searches an MRRS $r_*[1..i]$ among $|E(i)|+1$ candidates shown in Proposition 2 using already calculated $l(r_*[1..i'])$ for $i' < i$. Note that $h_*(i,j)$ can be calculated using already calculated $h_*(i-j,j)$, which can be accessed through the pointer to the entry for the repeat $s[i-3j+1..i-j]$. An MRRS $r_*[1..n]$ can be constructed by traceback if which one among $|E(i)|+1$ candidates is an MRRS $r_*[1..i]$ is memorized for

CMR-C %[Calculate an MRRS of a string for a Constant repetition weight function]
Input: $s[1..n]$: string,
w_Σ: alphabet weight function, w_R: constant repetition weight function
Output: $r_*[1..n]$: an MRRS representing $s[1..n]$

Step 1 Make the suffix tree T decorated with the endpoints of all primitive tandem repeats in the vocabulary for the reversed string of s using the algorithm developed by Gusfield and Stoye [2].
Step 2 Prepare linked list $E(i)$ for $i = 1, ..., n$ by executing Prepare_E(T's root node).
Step 3 For each entry of tandem repeat $s[i - 2j + 1..i]$ in $E(i)$ $(i = 1, 2, ..., n)$, set an MRRS and its size of $s[i - j + 1..i]$ to the entry by executing Set_Min_Size$(s[1..n], w_\Sigma, w_R, E)$.
Step 4 Calculate $r_*[1..n]$ by executing FindBestComb$(s[1..n], w_\Sigma, w_R, E)$.

Fig. 1. Algorithm CMR-C

each $i = 1, 2, ..n$. The time and space complexity of FindBestComb is $O(n \log n)$ because the number of occurrences of primitive tandem repeats is $O(n \log n)$.

Linked lists $E(i)$ for all $i = 1, ..., n$ and the pointers to the entries for tandem repeats $s[i-3j+1..i-j]$ which are set for all entries of primitive tandem repeat $s[i - 2j + 1..i]$ in $E(i)$ $(i = 1, ..., n)$ are calculated in $O(n \log n)$ time using the suffix tree of s decorated with the endpoints of all primitive tandem repeats in the vocabulary, which can be constructed in $O(n)$ time by the algorithm developed by Gusfield and Stoye [2]. The calculation can be done by postorder traversal of the decorated suffix tree for the *reversed string* of s. We call such a recursive algorithm Prepare_E. We also let each entry in $E(i)$ have a pointer to the entry for the leftmost occurrence of the same type tandem repeat, which is also set in algorithm Prepare_E. This pointer is used to save computational cost for occurrences of the same type tandem repeat as described below. Since the number of occurrences of primitive tandem repeats are $O(n \log n)$, the above procedure is done in $O(n \log n)$ time and $O(n \log n)$ space.

There is still one thing we have to do before executing FindBestComb for the whole string. An MRRS $r_*[i - j + 1, ..i]$ and its size $l(r_*[i - j + 1..i])$ in each entry for tandem repeat $s[i - 2j + 1..i]$ must be calculated. There are $O(n \log n)$ occurrences of primitive tandem repeats but MRRSs and their size have to be calculated only for distinct tandem repeats, namely, tandem repeats in the *vocabulary* (the set of different repeat types) [2], the number of which is $O(n)$. Applying FindBestComb to $s[i - j + 1..i]$ for each primitive tandem repeat $s[i - 2j + 1..i]$ in the vocabulary, an MRRS $r_*[i - j + 1..i]$ and its size $l(r_*[i - j + 1..i])$ can be calculated in $O(n \log n)$ time and $O(n \log n)$ space, so the minimum size for all tandem repeats in the vocabulary can be obtained in $O(n^2 \log n)$ time and $O(n \log n)$ space. We call the algorithm for this task Set_Min_Size.

Putting all together, algorithm CMR-C shown in Fig. 1, an algorithm finding an MRRS representing $s[1..n]$, is obtained.

By the argument so far, we have proved the following theorem.

Theorem 1. *$r_*[1..n]$ can be calculated in $O(n^2 \log n)$ time and $O(n \log n)$ space.*

If the number of occurrences of primitive tandem repeats is z, FindBestComb and Prepare_E can be executed in $O(n+z)$ time and $O(n+z)$ space. Furthermore, if the number of primitive tandem repeats in the vocabulary is w, Set_Min_Len can be executed in $O(w(n+z))$ time and $O(n+z)$ space. Thus, the following corollary holds.

Corollary 1. *$r_*[1..n]$ can be calculated in $O(w(n+z))$ time and $O(n+z)$ space when $w > 0$, where w and z is the number of primitive tandem repeats in the vocabulary and the number of their occurrences, respectively.*

4 Concluding Remarks

In this paper, we considered representations called an MRRS for a string, in which a set of disjoint or nested contiguous repeats are compacted, and proposed dynamic programming algorithms CMR and CMR-C to calculate it for a given string. CMR-C was theoretically proved to run in $O(w(n+z))$ time and $O(n+z)$ space, which indicates that it is fast if the number of distinct tandem repeat w and the number of its occurrences z are not large compared to the string length n. According to our experiments on DELL Precision T7500 (cpu: Intel(R) Xeon(R) E5520 [2.27GHz], memory: 2GB), CMC-R is fast enough for large-scale, synthetic datasets on strings; for a random binary string of 1,638,400 letters, 8.2 seconds were enough for CMR-C to find an MRRS. The size of MRRS can be a measure of how well a string is organized in terms of repeated structures. From our experimental result of such a structural complexity analysis applied to DNA sequences or chromosomes of the most major nine species (A. thaliana, C. elegans, zebrafish, fruit fly, chicken, human, mouse, yeast and rat), we found that the MRRS size was unique for each species.

Acknowledgements

This work was partially supported by JSPS KAKENHI 21500128 and the Collaborative Research Program of Institute for Chemical Research, Kyoto University (grant 2010-20).

References

1. Fraenkel, A. and Simpson, J.: The exact number of squares in Fibonacci words, *Theoretical Computer Science*, 218, pp.95-106, 1999.
2. Gusfield, D. and Stoye, J.: Linear time algorithms for finding and representing all the tandem repeats in a string, *Journal of Computer and System Sciences* 69, pp.525-546, 2004.
3. Main, M. and Lorentz, R.: An $O(n \log n)$ algorithm for finding all repetitions in a string, *Journal of Algorithms*, 5, pp.422-432, 1984.
4. Stoye, J. and Gusfield, D.: Simple and Flexible Detection of Contiguous Repeats Using a Suffix Tree, *Theoretical Computer Science*, 270, pp.843-856, 2002.