Linear Time Algorithms for Finding and Representing all Tandem Repeats in a String[1] [2]

Dan Gusfield and Jens Stoye

Computer Science Department
University of California, Davis.
CSE - 98 - 4

[1] An initial version of this technical report was circulated in August 1998. It differs from the present version primarily by the inclusion of Section 7.

[2] Research partially supported by grants DBI-9723346 from the NSF and DE-FG03-90ER60999 from the DOE, and the German Academic Exchange Service (DAAD).

Linear Time Algorithms for Finding and Representing all the Tandem Repeats in a String

Dan Gusfield[‡] Jens Stoye[§]

Abstract

A tandem repeat (or square) is a string $\alpha\alpha$, where α is a non-empty string. We present an $O(|S|)$-time algorithm that operates on the suffix tree $T(S)$ for a string S, finding and marking the endpoint in $T(S)$ of *every* tandem repeat that occurs in S. This decorated suffix tree implicitly represents all occurrences of tandem repeats in S, and can be used to efficiently solve many questions concerning tandem repeats and tandem arrays in S. This improves and generalizes several prior efforts to efficiently capture large subsets of tandem repeats.

1 Introduction

A tandem repeat (square) is a string of the form $\alpha\alpha$ where α is a non-empty string.

Given a string S of length n, a number of questions regarding tandem repeats may be asked. The simplest question is whether S contains a tandem repeat or is *squarefree*. Assuming a fixed alphabet size, this question is

[‡]Research partially supported by grant DBI-9723346 from the National Science Foundation, and by grant DE-FG03-90ER60999 from the Department of Energy. email: gusfield@cs.ucdavis.edu

[§]Research supported by the German Academic Exchange Service (DAAD). Present address: German Cancer Research Center (DKFZ), Abt. Theoretische Bioinformatik (H0300), Im Neuenheimer Feld 280, 69120 Heidelberg, Germany.

known be answerable in $O(n)$ time [Cro83, ML85, Cro86, CR94]. One might further be interested in identifying all *occurrences* of tandem repeats in S. Since there can be as many as $n^2/4$ occurrences of tandem repeats in a string of length n, an efficient algorithm for this task will depend on the output size, denoted z. Several $O(n \log n + z)$ time algorithms are known [ML84, LS93, SG98] for this task. It is often of interest to restrict the output to those tandem repeats which do not contain shorter repeats. These are called *primitive* tandem repeats. It is known that there can be at most $O(n \log n)$ occurrences of primitive tandem repeats in a string of length n, and several algorithms are known that identify those occurrences in $O(n \log n)$ time [Cro81, AP83, SG98].

In a very impressive, though highly technical, extended abstract, Kosaraju [Kos94] addresses the question of finding for each position i of S the *shortest* tandem repeat starting at position i, and sketches an $O(n)$ time algorithm for that problem. He also mentions the problem of finding all occurrences of primitive tandem repeats in S and, without details, states that the sketched algorithm can be extended to solve this question in $O(n + z)$ time where, as above, z is the size of the output.

Recently it was shown that the number of different *types* of tandem repeats contained in a string of length n is bounded by $O(n)$ [FS98]. Two tandem repeats $\alpha\alpha$ and $\alpha'\alpha'$ are of different type if and only if $\alpha \neq \alpha'$. Note that this $O(n)$ bound counts each tandem repeat type only once, no matter how many times that tandem repeat type occurs in the string. This $O(n)$ bound leads to an interesting challenge: Can we find one occurrence of each tandem repeat type in $O(n)$ time? Such a list of different repeat types (or more precisely a start location and length of each) is called the *vocabulary* (of tandem repeats) of string S. For example, a vocabulary of tandem repeats of the string `abaabaabbaaabaaba$` is given by the set of pairs $\{(1,6),(2,6),(3,2),(3,6),(8,2)\}$ representing the tandem repeats `abaaba`, `baabaa`, `aa`, `aabaab`, `bb`. The *set of occurrences* of tandem repeats on the other hand contains a pair for each occurrence of a tandem repeat in S. In the above example, the set of occurrences of tandem repeats is $\{(1,6),(2,6),(3,2),(3,6),(6,2),(8,2),(10,2),(11,2),(11,6),(12,6),(14,2)\}$.

Main Result

In this paper, we present an algorithm that finds the vocabulary of a string S of length n in $O(n)$ time and space. In so doing, the algorithm implicitly

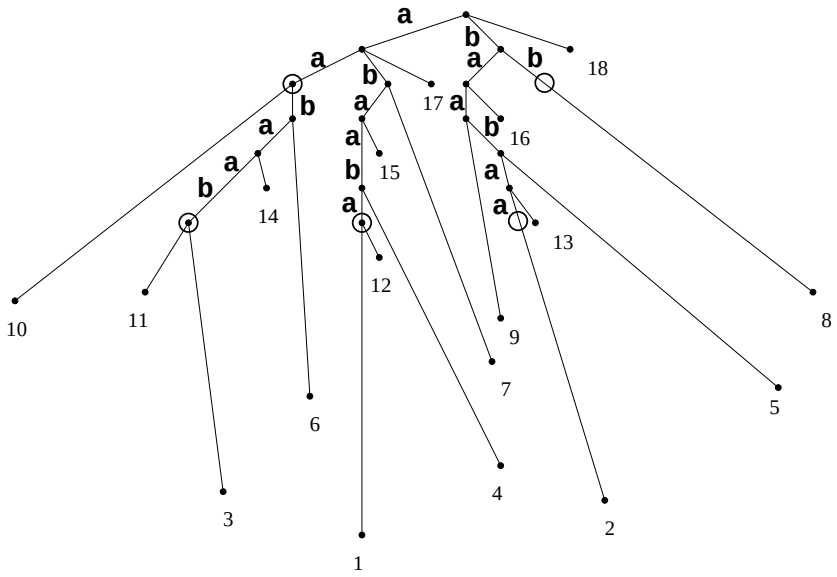

Figure 1: Suffix tree of string `abaabaabbaaabaaba$`. Circles indicate the endpoints of tandem repeats. Only the characters needed to spell out the tandem repeats are shown on the tree.

lays out the complete structure of the tandem repeats in S. The result is achieved in a three-phase procedure. Phase I finds a subset of the occurrences of tandem repeats, which we call a *leftmost covering set*, using an extension of Crochemore's linear-time algorithm that tests if S is squarefree [Cro83, Cro86, CR94], similar to the algorithm by Main [Mai89] which finds the leftmost occurrence of each tandem repeat type in $O(n)$ time. Phase II finds the end locations in the suffix tree of S for some of the tandem repeat types in the leftmost covering set. Phase III traverses parts of the suffix tree from the endpoints found in Phase II, to obtain the complete vocabulary of tandem repeats. The end result is that the suffix tree of S is *decorated* with the *endpoint* of each tandem repeat in the vocabulary of S. For an example, see Figure 1.

Clearly, such a decorated suffix tree compactly represents all the different tandem repeat types in S and the locations in S where they occur; it can be used to answer many questions regarding tandem repeats. For example, once the suffix tree has been decorated with the endpoint of each tandem repeat in the vocabulary, a standard linear-time traversal of the tree can identify for each location i in S, the shortest (and/or longest) tandem repeat that

begins at position i. Using the decorated suffix tree, all the above-mentioned problems concerning tandem repeats (and more) can be solved in time and space bounds that are as good or better than previously established.

Our $O(n)$-time method to decorate the suffix tree with the vocabulary of tandem repeats is based on ideas that are quite different from the ideas in [Kos94], and also different from those in [SG98].

Independent of our work presented here, a recent analysis of the number of possible *runs* of primitive tandem repeats was made, and a different linear-time algorithm was obtained [KK98] that finds all these runs in $O(n)$ time using an extension of Main's algorithm [Mai89]. Their algorithm then allows one to find all the z tandem repeat occurrences in S in $O(n + z)$ time, and in $O(n)$ space. However, their algorithm does not address the main result in our paper, finding the *vocabulary* of tandem repeats and locating them in a suffix tree in linear time, and we do not see how to extend their algorithm to achieve those goals.

2 Terminology and Technical Background

We assume a finite alphabet Σ of a fixed size. Throughout this paper we fix attention to a string S of length $n = |S|$. We assume S ends with a character '\$' not occurring elsewhere in S. For $1 \leq i \leq j \leq n$, $S[i..j]$ denotes the substring of S beginning with the ith and ending with the jth character of S; we say there is an *occurrence* of $S[i..j]$ at position i in S. String wa is called the *right-rotation* of string aw, where a is a single character and w is a non-empty substring.

A string $w \in \Sigma^+$ is a *tandem repeat* if it can be written as $w = \alpha\alpha$ for some $\alpha \in \Sigma^+$. An occurrence of a tandem repeat $\alpha\alpha = S[i..i + l - 1]$ is represented by a pair (i, l), called a *tandem repeat pair*. The first entry of a tandem repeat pair is called the *position entry*, and the second entry is called the *length entry*. Two occurrences of tandem repeats $S[i..i + l - 1] = \alpha\alpha$ and $S[i'..i' + l - 1] = \alpha'\alpha'$ are of the same *type* if and only if $\alpha = \alpha'$. For simplicity, we will sometimes specify a tandem repeat type by referring to an occurrence (i, l) of that repeat type, even though the specific location is not of interest. The *vocabulary* (of tandem repeats) of S is a set of tandem repeat pairs such that each type of tandem repeat occurring in S is contained in the set exactly once. In contrast, the *set of occurrences* (of tandem repeats) of S contains all the tandem repeat pairs of S.

An interval of positions $i, i+1, \ldots, j$ is called a *run of l-length tandem repeats* if $(i,l), (i+1,l), \ldots, (j,l)$ are each tandem repeat pairs. A tandem repeat pair (i,l) *covers* another tandem repeat pair (j,l) if and only if there is a run of l-length tandem repeats in S that starts at i and contains j.

Note that if (i,l) covers (j,l), then the substring $S[j..j+l-1]$ can be obtained by a series of successive right-rotations from the substring $S[i..i+l-1]$, and by definition, each string created by a right-rotation is also a tandem repeat of length l. In our example string `abaabaabbaaabaaba$`, the tandem repeat pair $(1,6)$ covers the pairs $(2,6)$ and $(3,6)$, the pair $(10,2)$ covers $(11,2)$, and $(11,6)$ covers $(12,6)$.

A set of pairs P is called a *covering set* if and only if at least one occurrence of every repeat in the vocabulary of tandem repeats is covered by one of the pairs in P. That is, the runs starting from every pair in P collectively cover at least one occurrence of each tandem repeat type. A set of pairs P is a *leftmost covering set* if the leftmost occurrence of each type of tandem repeat in S is covered by a pair. For example, $\{(1,6),(8,2),(11,2)\}$ is a covering set of `abaabaabbaaabaaba$`, but is not a leftmost covering set since the leftmost occurrence of `aa` at position 3 is not covered. However, $\{(1,6),(3,2),(8,2)\}$ is a leftmost covering set.

2.1 The Size of the Vocabulary

The following result by Fraenkel and Simpson [FS98] is essential to our present work.

Theorem 1. For any position i in S, there can be at most two tandem repeat types whose rightmost occurrences start at position i. Stated differently, even though there may be many tandem repeat types that occur starting at position i, all but two (at most) of these types will also occur starting somewhere to the right of i.

Corollary 1. The size of the vocabulary of tandem repeats of any string of length n is bounded by $2n$.

Actually, Fraenkel and Simpson give slightly tighter bounds for the size of a tandem repeat vocabulary. They establish that for strings of length $n \geq 5$ the size of the vocabulary is bounded by $2n-8$, and that for binary strings of length $n \geq 22$, it is bounded by $2n-29$.

2.2 Background on Suffix Trees

We use suffix trees extensively both as computation tools, and as the data structure that holds the output of the computation. For a general introduction to suffix trees, see [CR94] or [Gus97].

We use $T(S)$ to denote the suffix tree of S, i.e., the compacted trie of all the suffixes of S; $L(v)$ denotes the *path-label* of a node v i.e., the concatenation of the edge labels along the path from the root to v. We say that v is *path-labeled* $L(v)$. $D(v) = |L(v)|$ is the *string-depth* of v. Each leaf v of $T(S)$ is also labeled with index i if and only if $L(v) = S[i..n]$. For a non-empty substring $w \in \Sigma^+$ of S, we encode the *endpoint* of w in $T(S)$ by a pair (e, m), where $e = (u, v)$ is an edge in $T(S)$ and m is an integer satisfying $0 < m \leq D(v) - D(u)$. The meaning is that string w ends m characters from u (u is the parent of v) along the edge (u, v). Clearly, $w = L(u)\beta$ where β is the m-length prefix of the edge-label of e. Note that w is encoded on an edge (u, v) even if its endpoint is at node v.

The *suffix link* of a node v with label $L(v) = aw$, $a \in \Sigma$, $w \in \Sigma^*$, is a pointer to the node labeled w. This node always exists if v is a non-root internal node of $T(S)$. The character a is the *label* of the suffix link pointing from the node labeled aw to the node labeled w. It is well known that the suffix tree of S including the suffix links can be computed in $O(n)$ time [Wei73, McC76, Ukk95, Far97].

3 Phase I: Finding a Leftmost Covering Set

Crochemore [Cro83, Cro86, CR94] developed a linear-time algorithm that determines if a string is squarefree. In this section we show how this algorithm can be extended to find a leftmost covering set of the tandem repeats of S. That is the key task of Phase I. Crochemore's algorithm, and our modification of it, use two crucial tools. The first is a decomposition of the string S, called the *Lempel-Ziv* (LZ) decomposition, and the second is the repeated use of *longest common extension queries*. We first describe the LZ decomposition of S [LZ76].

For each position i of S, let l_i denote the length of the longest prefix of $S[i..n]$ that also occurs as a substring of S starting at some position $j < i$; let s_i denote the starting position of the leftmost occurrence of this substring in S if $l_i > 0$, and $s_i = 0$, otherwise. The *Lempel-Ziv (LZ) decomposition*

text	a	b	a	a	b	a	a	b	b	a	a	a	b	a	a	b	a	$
position i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
l_i	0	0	1	5	4	3	2	1	3	2	6	6	5	4	3	2	1	0
s_i	0	0	1	1	2	3	1	2	2	3	3	1	2	3	1	2	1	0

Table 1: The definition of l_i and s_i.

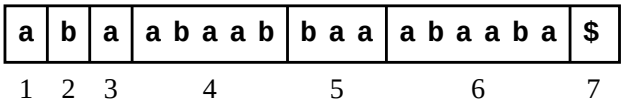

Figure 2: The Lempel-Ziv decomposition.

of S is the list of indices $i_1, i_2, \ldots, i_k$, defined inductively by $i_1 = 1$ and $i_{B+1} = i_B + \max(1, l_{i_B})$ for $i_B \leq n$. The substring $S[i_B..i_{B+1}-1], 1 \leq B \leq k$, obtained in this way is called the Bth *block* of the LZ decomposition of S. It is well known that this decomposition can be computed in $O(n)$ time, e.g. using the suffix tree of S [RPE81] [1]. Table 1 shows the values s_i and l_i for the string `abaabaabbaaabaaba$`, and Figure 2 shows the Lempel-Ziv decomposition.

The following two basic facts are stated explicitly or implicitly in [Cro83, Cro86, CR94] and connect tandem repeats to the blocks of the LZ decomposition.

Lemma 1. The right half of any tandem repeat occurrence must touch at most two blocks of the LZ decomposition.

Proof. Assume the scenario shown in Figure 3, where the right half of tandem repeat $\alpha\alpha$ touches more than two blocks. Let β be the first block completely included in the right half of the tandem repeat, and let γ be the remaining (necessarily non-empty) suffix of the right α. Since $\alpha\alpha$ is a tandem repeat, there is an earlier occurrence of $\beta\gamma$ in S, namely as a suffix of the first half of the tandem repeat. But then the second β is not maximal with

[1] There are several published variants of this decomposition, with differing names. Crochemore [CR94] uses the variant defined above, but calls it an *f-factorization*, and reserves the names Ziv and Lempel for a related decomposition. That decomposition is called Ziv-Lempel in [Gus97], where its efficient computation is detailed. Those details easily extend to the LZ decomposition defined above.

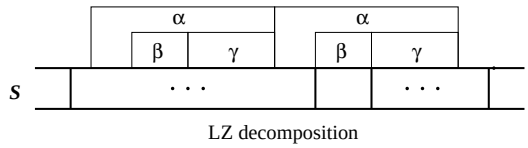

Figure 3: The right side of a tandem repeat must touch at most two blocks of the Lempel-Ziv decomposition.

respect to having appeared earlier, contradicting the assumption that the second β is a full block. □

Lemma 2. The leftmost occurrence of any tandem repeat must touch at least two blocks.

Proof. By the definition of the LZ decomposition, any substring which occurs completely contained in one block has an occurrence starting at an earlier position in S. Hence it can not be the leftmost occurrence of that substring. □

We say that the *center* of a tandem repeat $\alpha\alpha$ is *inside* block B if the rightmost character of the left copy of α is contained in B. The prior two Lemmas establish the following:

Theorem 2. If the leftmost occurrence of a tandem repeat $\alpha\alpha$ has its center inside some block B, then either

> (Condition 1) $\alpha\alpha$ has its left end inside block B and its right end inside block $B+1$;

or

> (Condition 2) the left end of $\alpha\alpha$ extends into block $B-1$ and possibly further left. (The right end may be inside block B or inside block $B+1$.)

See Figure 4. □

It follows immediately that in order to identify the leftmost occurrence of each type of tandem repeat, it suffices to look only for occurrences of tandem

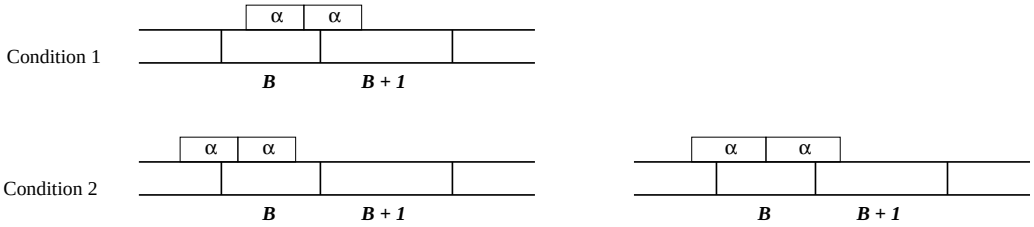

Figure 4: The two conditions for leftmost tandem repeats.

repeats that satisfy one of the two above conditions. This point is crucial in the linear time bound for finding a leftmost covering set.

We now describe the second crucial tool of Phase I, the use of the *longest common extension query*.

For two indices i and j of a string S, we define the *longest common extension* from i and j (in the forward direction) as the length of the longest substring in S that *starts* at i and matches a substring starting at j. We define the longest common extension in the backward direction as the longest substring of S that *ends* at i and matches a substring ending at j. It is known that after linear processing time of S, any longest common extension computation (forward or backward) can be executed in constant time. This is achieved in one of two ways, either by using a constant time least common ancestor algorithm, or more simply by using variants of Knuth-Morris-Pratt or Boyer-Moore or Z-algorithm preprocessing. See [Gus97] pages 196 and 208 for more details on these two approaches. We assume that string S has been preprocessed so that any subsequent longest common extension query can be executed in constant time.

With the LZ decomposition, and the use of longest common extension queries, we can now begin to develop the Phase I algorithm that finds a leftmost covering set. The algorithm is an extension of Crochemore's algorithm for determining if a string is squarefree. The algorithm processes blocks $1, 2, \ldots$ of the LZ decomposition in order, and it outputs an ordered list of tandem repeat pairs as each block is processed. The algorithm maintains the invariant that after processing blocks $1, \ldots, B$ of the LZ decomposition, all occurrences of tandem repeats whose center is inside some block B' from 1 to B, and that satisfy either Condition 1 or Condition 2, will be covered by the pairs output by the algorithm. Since the leftmost occurrence of any tandem repeat satisfies either Condition 1 or 2, the algorithm will have output

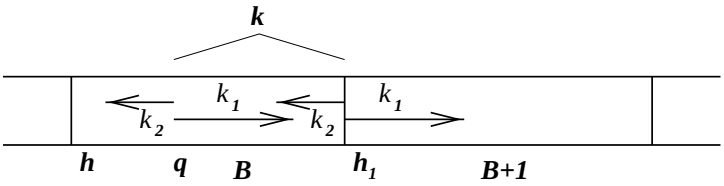

Figure 5: Algorithm 1a.

a leftmost covering set at termination. Every block B is processed with two algorithms that are described next.

3.1 Algorithm 1: Processing block B

Assume block B starts at position h and block $B+1$ starts at position h_1.

Algorithm 1a: processing block B for tandem repeats that satisfy Condition 1 (see Figure 5)

For k from 1 to $|B|$ do
begin

Let $q = h_1 - k$.

Compute the longest common extension in the *forward* direction from positions h_1 and q. Let k_1 denote the length of that extension.

Compute the longest common extension in the *backward* direction from positions $h_1 - 1$ and $q - 1$. Let k_2 denote the length of that extension.

If $k_1 + k_2 \geq k$ and $k_1 > 0$, then output the tandem repeat pair $(\max(q - k_2, q - k + 1), 2k)$.

end.

The key to understanding Algorithm 1a is the fact that there is a run of tandem repeats, each of length $2k$, starting at position $q - k_2$ that cover position $q - 1$ at least, if and only if $k_1 + k_2 \geq k$. That claim is easily established and left to the reader. If $k_1 > 0$ then that run also extends into block $B+1$, and hence contains tandem repeats of length $2k$ which have their center in B and their right end in $B + 1$. In that case, we want to output a repeat pair that covers those tandem repeats. The pair $(q - k_2, 2k)$ does that, but if $k_2 \geq k$, the pair $(q - k + 1, 2k)$ also covers all the desired tandem repeats, and in that case we will use $(q - k + 1, 2k)$ to simplify the exposition of a subsequent computation.

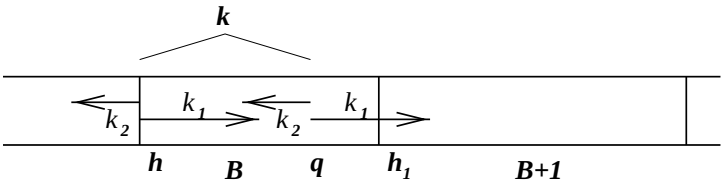

Figure 6: Algorithm 1b.

Algorithm 1b: processing block B for tandem repeats that satisfy Condition 2 (see Figure 6)

For k from 1 to $|B| + |B+1|$ do
begin

Let $q = h + k$.
Compute the longest common extension in the *forward* direction from positions h and q. Let k_1 denote the length of that extension.
Compute the longest common extension in the *backward* direction from positions $h-1$ and $q-1$. Let k_2 denote the length of that extension.
If $k_1 + k_2 \geq k$ and both k_1 and k_2 are greater than 0, and $\max(h - k_2, h - k + 1) + k < h_1$, then output the tandem repeat pair $(\max(h - k_2, h - k + 1), 2k)$.

end.

The explanation of Algorithm 1b is similar to that of Algorithm 1a. The point is to output a tandem repeat pair that covers all the length k tandem repeats whose center is in B and whose left end is in a block to the left of B. Details are again left to the reader. Algorithms 1a and 1b are together referred to as Algorithm 1. Figure 7 continues our example.

Theorem 3. Algorithm 1 outputs a leftmost covering set of pairs in $O(n)$ time and space.

Proof. We have already established that Algorithm 1 outputs a leftmost covering set of tandem repeat pairs. Algorithm 1a runs in $O(n)$ time because each block B is processed in $O(|B|)$ time, and the blocks are disjoint. In Algorithm 1b, each block B is processed in time proportional to $|B| + |B+1|$. Hence over the entire running of Algorithm 1b, each block (except the first and last) contributes twice to the total count. But since the blocks are disjoint, the total time for Algorithm 1b is also $O(n)$. □

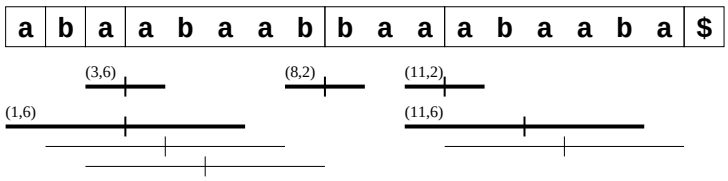

Figure 7: The result of Algorithm 1 on string `abaabaabbaaabaaba$`. The tandem repeats from the leftmost covering set are shown as thicker lines below the string. The corresponding tandem-repeat pairs (i, l) are written next to them. $(1, 6)$ is output by Algorithm 1b while processing block three; $(3, 6)$ is output by Algorithm 1a while processing block three; $(8, 2)$ is output by Algorithm 1a on block four; $(11, 2)$ is output by Algorithm 1a on block five; and $(11, 6)$ is output by Algorithm 1b on block six.

For each position i in S, let $P(i)$ denote the list of pairs (i, l^i_j) output by Algorithm 1, which have position entry i. Let P denote the complete set of the output pairs, i.e., the union of the $P(i)$. One additional implementation detail is needed in Phase I, in order to facilitate the work in Phase II.

Lemma 3. Without increasing the worst-case running time of Algorithm 1, the lists $P(i)$ can be accumulated so that for each position i, the pairs (i, l^i_j) in $P(i)$ are sorted by decreasing order of the length entry. That is, $l^i_j > l^i_{j+1}$.

Proof. This is achieved by attaching an initially empty list $P(i)$ to each position i of S. Then as each pair (i, l) is output by Algorithm 1, the pair (i, l) is pre-pended to the list $P(i)$. To see that this gives the desired result, focus on a fixed pair (i, l) output while block B is processed. We claim that at the time (i, l) is output, all the pairs (i, l') with $l' < l$, have already been added to list $P(i)$. First consider all pairs (i, l') with center in blocks $1, \ldots, B-1$. These pairs were output in earlier iterations of Algorithm 1, and hence are already in $P(i)$. Moreover, $l' < l$ because l' (respectively l) is the distance between i and the center of the tandem repeat (i, l') (respectively (i, l)).

Second, when processing block B, Algorithms 1a and 1b do not output pairs with the same i value. The reason is that Algorithm 1a outputs pairs whose i falls in block B, while Algorithm 1b outputs pairs whose i falls in blocks before B.

Finally, Algorithm 1a (and 1b) vary k in increasing order, and hence if (i, l') and (i, l) are output in that order when Algorithm 1a (or 1b) is processing block B, then $l' < l$.

For the time bound, note that because the tandem repeat pairs are output in $O(n)$ time and each entry in a list is pre-pended to the list in constant time, the ordered lists are accumulated in $O(n)$ time as well.

□

4 Phase II: Marking the endpoints of some tandem repeat types

Each tandem repeat pair in the leftmost covering set P found in Phase I specifies a particular tandem repeat occurrence, and hence a particular tandem repeat type. (Note we are not talking about the repeats covered by a repeat pair, but only the single repeat specified by the pair itself.) Let Q denote the set of tandem repeat types specified by P. To introduce the idea of Phase II, we motivate it by saying that we would ideally like to mark the endpoints of the repeats in Q during Phase II. However, because of time constraints, that will not be possible, and Phase II will only mark the endpoints of a particular *subset* Q' of Q. The full explanation for the use of Q' will have to wait until part of Phase III is introduced.

Phase II processes every non-root node of $T(S)$ during a linear-time, *bottom-up* traversal. To start, each leaf i is given the list $P(i)$ computed in Phase I, and Q' is the empty set. During the traversal, each internal node v will be given some end-portion of a list given to one of its children (details below). The list given to v will be denoted $P(v)$; by induction and Lemma 3, $P(v)$ is guaranteed to be sorted by decreasing order of its length entries. The algorithm processes each node v (which could be a leaf), whose parent is denoted u, as follows:

Repeat

Let (i, l) denote the pair at the head of the list $P(v)$;
If $l > D(u)$ {Tandem repeat (i, l) ends at node v or on the interior of the edge (u, v).}
Then begin
Store the number $l - D(u)$ on edge (u, v) to record that (i, l) ends $l - D(u)$ characters from u along edge (u, v);
Remove (i, l) from $P(v)$; {For the exposition, place pair (i, l) into list P'.}

end;
Until either $P(v)$ is exhausted, or $l \leq D(u)$.

Since the traversal is bottom-up, a node u is processed only after all of its children have been processed. When u is processed it would be appealing to merge-sort the current lists of the children of u, according to the length entries of the pairs, and give that merged list to u. If that were done for each node u, then the algorithm would record the endpoint of each repeat in Q. However, over the entire tree, that approach would take $\Omega(n^2)$ time for the merges, even though the size of all the lists is $O(n)$. But, as will be proved in the next section, merging can be avoided as follows.

Before the traversal, label each node with the smallest leaf number in its subtree. Note that the label at each node u agrees with the label of exactly one of its children. Then during the bottom-up traversal, after all the children of an internal node u have been processed, set the list $P(u)$, given to u, to the current list $P(v)$, where v is the child of u with the smallest label. The traversal now takes $O(n)$ time, since the size of the original lists is $O(n)$, and no lists are merged. Moreover, if node u is labeled with leaf i, then the act of creating and "giving" a list to u is implemented by passing a pointer to u that points to the appropriate position in list $P(i)$. Hence Phase II takes $O(n)$ time.

Let Q' be the set of distinct tandem repeat types specified by the pairs in P' at the end of Phase II. Clearly, Q' is the set of tandem repeat types that are recorded in $T(S)$, and Q' is a subset of Q and might be a strict subset. The utility of Q' will be explained in the next section.

5 Phase III: Using Q' to record the endpoints of the full vocabulary

The end result of Phase II is the decoration of the suffix tree with the endpoints of the tandem repeat types in Q'. We will now use those endpoints to find the endpoints of all the tandem repeats in the full vocabulary of tandem repeats. The key algorithmic operation is that of a *suffix-link walk*, which is a common operation in building and using suffix trees.

Consider a tandem repeat $\alpha\alpha = a\gamma$, where a is a single character. A *suffix-link walk* (or *walk* for short) from the end of $\alpha\alpha$ first moves to a location

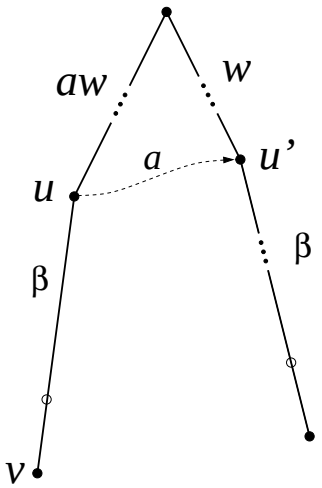

Figure 8: The definition of a suffix link walk.

in $T(S)$ labeled with the string γ. There are two cases to consider. In the first case, the end-location in $T(S)$ of $\alpha\alpha$ is at a node v. In this case, the suffix-link walk first moves along the suffix link (v, v') to the node v' labeled γ. In the second case, the endpoint of $\alpha\alpha$ is strictly in the interior of an edge (u, v) entering v. In that case, denote $Q(u)$ by aw, and let β denote the (non-empty) prefix of the label of edge (u, v) such that $\gamma = w\beta$ (see Figure 8). In this case, the walk starting at the end of $\alpha\alpha$ first moves to node u, and then follows the suffix link of node u to a node denoted u'. By construction, the suffix link (u, u') is labeled a, and the node u' is labeled w. From u', the suffix-link walk follows the path labeled β (which always exists) until it reaches the end-location of string $\gamma = w\beta$.

When the suffix-link walk reaches the end of string γ, in either of the two cases, the algorithm tests if there is some path from the endpoint of γ that starts with the character a (the starting character of $\alpha\alpha$). If that continuation exists, the algorithm moves to the endpoint of string γa, and the suffix-link walk is called *successful*; otherwise the walk terminates at the endpoint of γ and is called *unsuccessful*. A successful walk from $a\gamma$ corresponds to a right-rotation of $a\gamma$, creating the tandem repeat γa. Hence a run of right-rotations in S defines a *chain* of successful walks, each walk starting where the previous one ended.

For efficiency, one implementation detail is needed in the second case. When a walk follows the path labeled β from node u', it can traverse each

edge e on the path in constant time, no matter how long the label of e is. This is accomplished by using the skip/count trick that is standard in many suffix tree algorithms: since a path labeled β must extend from u', any edge whose label is shorter than (the remainder of) β can be skipped in one step, and the walk continued at the next node with the appropriately truncated suffix of β. Determining which edge to traverse from any encountered node simply requires finding the unique edge whose label starts with the correct next character of β. For more details on the skip/count trick see [Gus97].

Now that the suffix-link walk has been defined, we can more fully explain the utility of Q'. We define a set Q^* of tandem repeat types to be *sufficient* if the endpoint in $T(S)$ of every tandem repeat in the vocabulary of S can be reached by some chain of successful suffix-link walks starting from the endpoint in $T(S)$ of some tandem repeat type in Q^*.

Theorem 4. The subset Q' of Q defined during Phase II is a sufficient set of tandem repeat types.

Proof. By the definitions of a chain of walks and a run (of right rotations), if the pair (i, l) covers the pair (j, l) in S, then a chain of successful walks in $T(S)$ starting at the endpoint of the tandem repeat (i, l) must reach the endpoint of the tandem repeat (j, l). It follows that Q is sufficient, since P is a (leftmost) covering set. Moreover, by transitivity, if the endpoint of every tandem repeat type in Q is reached by a chain of walks from the endpoint of some tandem repeat type in Q', then Q' is also sufficient.

Define Q'' to be the set of tandem repeat types in Q whose endpoints in $T(S)$ are *not* reached by any chain of successful walks from Q'. To prove the theorem, assume for contradiction, that $Q'' \neq \emptyset$. Let $P'' \subset P$ be the set of pairs in P which specify tandem repeat types in Q''. Let (j, l) be a repeat pair in P'' such that j is smaller or equal to the position entry of any pair in P''. Let $\alpha\alpha$ denote the tandem repeat type specified by (j, l).

Clearly, if j were the leftmost starting position of $\alpha\alpha$ in S then (j, l) could never have been removed in Phase II, so (j, l) would be in P' and not in P''. So, the leftmost occurrence of $\alpha\alpha$ must start at some position $q < j$. Now because P is a leftmost covering set, there is some pair $(r, l) \in P$ which covers (q, l). But $r \leq q < j$, so (r, l) is in P' by the choice of j. Hence a chain of walks in $T(S)$ from the endpoint of the tandem repeat specified by

(r, l) must reach the endpoint of $\alpha\alpha$ in $T(S)$, a contradiction. We conclude that Q'' is empty, and Q' is sufficient. □

Now we are ready to present the Phase III algorithm which decorates the suffix tree. Recall that an endpoint of a tandem repeat in Q' is recorded in $T(S)$ on edge (u, v) if the repeat ends at node v or in the interior of edge (u, v). We will continue to use this convention to record the endpoints of additional tandem repeats that we find. However, we distinguish between tandem repeats in Q' and the new ones found in Phase III.

At the high level, Phase III executes a linear-time (depth-first say) traversal of $T(S)$. This traversal is interwoven with suffix-link walks. In detail, when the traversal encounters a tandem repeat $\alpha\alpha$ in Q' recorded at some edge (u, v), it executes a chain of suffix-link walks in $T(S)$ (starting from the end of $\alpha\alpha$), to find and record in $T(S)$ all of the right rotations of $\alpha\alpha$ which are tandem repeats. This chain of walks ends the first time an unsuccessful walk ends, or the first time that a successful walk ends at the endpoint of a tandem repeat that has already been recorded in $T(S)$. After terminating this chain of walks, the algorithm returns to edge (u, v). It executes one chain of walks for each tandem repeat in Q' recorded on edge (u, v). After all of these walks have been executed, the algorithm continues its linear-time traversal of $T(S)$ from edge (u, v). Some additional implementation details will be introduced when the time analysis is considered.

In the example shown in Figure 9, the pair $(11, 6)$ (representing tandem repeat `aabaab`) is in list $P(11)$, but `aabaab` is not in Q' (because of leaf 3). However, the endpoint of tandem repeat `abaaba` is in Q', and two right-rotations of it create `aabaab`. Hence the endpoint of `aabaab` is reached in Phase III after two successful suffix-link walks.

5.1 Analysis of Phase III

5.1.1 Correctness

Theorem 5. Phase III correctly records the endpoint in $T(S)$ of each tandem repeat in the vocabulary of S.

Proof. Recall that Q' is sufficient, and a run of right-rotations from a pair (i, l) corresponds to a chain of successful suffix-link walks that start at the endpoint of the tandem repeat defined by (i, l). Hence, if every chain of suffix

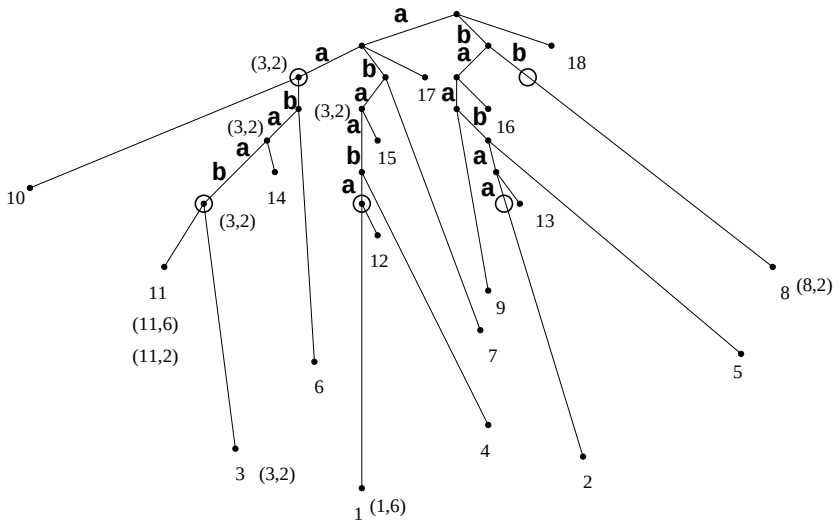

Figure 9: Example (cont.)

link walks from endpoints of tandem repeats in Q' ended only at the end of an *un*successful walk, then correctness of Phase III would be immediate. However, a chain of suffix-link walks may also end at the endpoint of a tandem repeat that is already recorded in $T(S)$. If that ending tandem repeat is guaranteed to be in Q', then correctness is again immediate, since a chain of walks will be (or has been) started from the end of that tandem repeat.

We will show that every chain of walks in Phase III ends either with an unsuccessful walk, or at the endpoint of a tadem repeat in Q'. For contradiction, suppose two chains of successful walks start at the endpoints of two tandem repeats in Q', $\alpha\alpha$ and $\alpha'\alpha'$ respectively, and that both chains contain successful walks that end with the same tandem repeat $\alpha''\alpha''$ not in Q'. Suppose one of those chains first finds and records the end of $\alpha''\alpha''$, and the other later encounters this record. Suppose the chain from $\alpha\alpha$ contains more walks than the other chain does.

A successful walk in $T(S)$ corresponds to a right rotation of a specific string. Hence, two successful walks that end at the same point in $T(S)$ must have also started at the same point. Repeating this reasoning, the chain from $\alpha\alpha$ must contain the chain from $\alpha'\alpha'$. But this is not possible in Phase III, for then the chain from $\alpha\alpha$ would have ended with the walk that ends at the endpoint in $T(S)$ of string $\alpha'\alpha'$. We conclude that every chain of walks

in Phase III ends either with an unsuccessful walk, or at the endpoint of a tandem repeat in Q', and the correctness of Phase III is proven. □

5.1.2 Time and Space Analysis of Phase III

We begin with two Lemmas.

Lemma 4. For any edge $e = (u, v)$ in $T(S)$, at most two tandem repeat types end in the interior of edge e or at node v.

Proof. For each node v of $T(S)$, let $R(v)$ denote the largest of all leaf-labels in the subtree below v. Then the rightmost occurrence of a substring of S whose endpoint ends in the edge $e = (u, v)$ starts at position $i = R(v)$ in S. Since there can be at most two tandem repeat types whose rightmost occurrences start at the same position i (see Theorem 1), the number of tandem repeats whose endpoints are contained in edges which end at nodes with the same R-value is bounded by two. In particular this means, that there can be at most two tandem repeats whose endpoints are contained in edge (u, v) or at node v. □

Lemma 5. The total number of edges traversed during all the suffix-link walks in Phase III is bounded by $O(|\Sigma|n)$.

Proof. By Corollary 1, the number of different tandem repeat types in S is bounded by $2n$. Each suffix-link walk begins at the endpoint of a distinct tandem repeat type, no walk is repeated twice. Therefore, the total number of suffix-link walks, and hence suffix-link traversals, is bounded by $2n$. We next bound the total number of edges skipped by the skip/count trick.

The suffix-link walk from a tandem repeat whose endpoint is at a node involves no edge skipping, hence edges are skipped only when the starting point of a walk is in the interior of an edge. Before analyzing the time for these skips, consider an edge (u, v) labeled with string β. There is a single suffix link, labeled a say, from u to u', and a single suffix link from v to v', also labeled a. By construction, v' must be a descendant of u', and the path from u' to v' must be labeled with the string β. Therefore, any suffix-link walk that starts on the edge (u, v) must end on the path from u' to v'. Again by construction, there can be no node strictly between u' and v' that has an

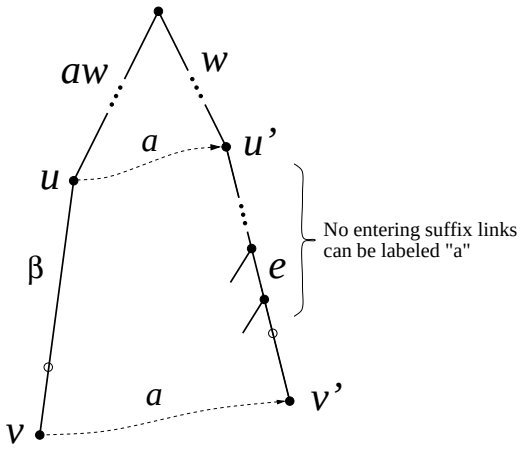

Figure 10: All the suffix-link walks that skip edge e and traverse a suffix link labeled a must start on the same edge, or at the head of that edge.

incoming suffix-link labeled with a. It follows that any suffix-link walk that traverses a suffix link labeled a, must end before encountering a second node that has an entering suffix link labeled a. We conclude that if e is an edge on the path from u' to v', and e is skipped during a suffix-link walk which started by traversing a suffix link labeled a, then that walk must have started on edge (u, v). See Figure 10.

What we have established is that if e is skipped during a suffix-link walk, then the label of the suffix link used on that walk uniquely determines a single edge where that walk could have begun. But every suffix-link walk starts at the endpoint of a tandem repeat, so by Lemma 4, at most two suffix-link walks can start on any given edge. Hence the number of times that edge e can be skipped (each time during a different walk) is bounded by $2|\Sigma|$, and the lemma is proved. □

Each suffix-link traversal, or edge traversal, or edge skip in a walk requires only constant time. Hence to finish the time analysis of Phase III we only need to bound the time needed to test if a new walk should be started after a successful walk has ended. Suppose a successful suffix-link walk begins at the end of a tandem repeat of length l. It ends at string-depth l also, so to implement the test, the algorithm checks to see if a tandem repeat of length l is recorded on the appropriate edge (the edge into a node if the walk ends at a node, else on the edge where the walk ends). The time to check for such a record is constant since, by Lemma 4, at most two tandem repeats can be

recorded on any edge. Finally, the only additional space needed in Phase III is used to record the endpoints of the tandem repeats in $T(S)$. Since each such record takes constant space, and there are only a linear number of them, we conclude with the following

Theorem 6. Phase III runs in $O(n)$ time and space, and when finished, the endpoints of all tandem repeat types are recorded in $T(S)$. Hence $T(S)$ can be decorated with the complete tandem repeat vocabulary of S in $O(n)$ time and space.

Corollary 2. By a linear time traversal of $T(S)$, the complete vocabulary of the tandem repeats of S can be collected and output (as position, length pairs) in $O(n)$ time and space.

6 Extensions of the Basic Algorithm

6.1 Many immediate extensions

Once the suffix tree $T(S)$ is decorated with the endpoints of all tandem repeats, several questions regarding tandem repeats in S can easily be answered. In this section we mention some of these.

Certainly, all occurrences of tandem repeats in S can be found if for each tandem repeat location, the subtree below this location is traversed and the labels of all leaves in this subtree are reported. The space required for this algorithm is only $O(n)$, and since each subtree traversal is possible in time proportional to the number of its leaves, the total running time of this extension is $O(n + z)$, where z is the output size[2]. Note that existing algorithms that find all occurrences of tandem repeats and run in $O(n \log n + z)$ time were previously declared (in some places) to be time-optimal, because

[2]We should note that Algorithm 1 can be extended to directly find all occurrences of tandem repeats in this time bound, using $\Omega(n + z)$ space. The key is to pick up and copy those tandem repeats that are entirely contained in a single block, since all others are collected in the original Algorithm 1. We leave the details to the reader. Crochemore's original algorithm [Cro83, Cro86, CR94] that tests if a string is squarefree, can also be extended to find either the leftmost occurrence of each tandem repeat in $O(n)$ time and space [Mai89], or to find all the runs of tandem repeats in $O(n)$ time and space, [KK98], and to find all occurrences of tandem repeats in $O(n + z)$ time and $\emptyset(n)$ space [KK98].

in worst-case there can be $\Omega(n^2)$ occurrences. Hence our new algorithm with its $O(n + z)$ time bound is "more optimal" than the previous "optimal" algorithms.

Several questions can be answered in $O(n)$ time by propagating information about the tandem repeats down towards the leaves of the suffix tree. These questions include finding the number of tandem repeats starting at each position of S (and hence the total number of tandem repeats in S), and finding the shortest or the longest tandem repeat starting at each position of S. The details are left to the reader. More complex extensions are discussed in the next subsection, and in Section 7.

6.2 Primitive Tandem Repeats

Recall from the introduction that a *primitive* tandem repeat is a tandem repeat $\alpha\alpha$ where α is primitive, i.e., $\alpha = \beta^k$ for some non-empty string β only if $k = 1$.

We decorate the suffix tree with the endpoints of all the primitive tandem repeat types in S by filtering out non-primitive repeats from the suffix tree decorated with the complete repeat vocabulary. We use an auxiliary data structure *depth-array* of size n, where all cells are initialized to zero.

Tree $T(S)$ is traversed in a depth-first order. Each time an endpoint of a tandem repeat of length l, say, is encountered, we do the following. If the entry of cell l in the depth-array is 0, we insert the value l into cell $2l$ of the depth-array. If the entry of cell l is $k \neq 0$, we insert the value k into cell $l + k$ of the depth-array, and we mark the tandem repeat we just encountered as non-primitive. When the depth-first traversal backs up to the endpoint of a tandem repeat of depth l (primitive or not), it sets the depth-array value for cell l to zero.

After the depth-first traversal, every non-primitive tandem repeat type in the vocabulary is marked in $T(S)$, so the primitive tandem repeat types can be collected in $O(n)$ time with another linear-time tree traversal. Further, all the occurrences of primitive tandem repeats can be found in optimal $O(n+z)$ time, where z is now the number of occurrences of primitive tandem repeats in S. This algorithm is again "more optimal" than the previous $O(n \log n)$ time algorithms.

7 Tandem Arrays

A *tandem array* is a string $w = \alpha^k$ with $k \geq 2$. If α is primitive, w is a *primitive tandem array*. Generalizing the goal of finding the vocabulary of all tandem repeat types, we would like to find the set of all distinct tandem array types or (more often) the distinct primitive tandem array types in a string. We don't know the size of those sets, but assuming there are z distinct primitive tandem array types in the string, we can find one representative of each type in $O(n + z)$ time. The method again relies heavily on the use of suffix trees.

The algorithm works in two phases. To explain the first phase, we define a set Q of primitive tandem *repeat* types to be *p-sufficient* if the endpoint in $T(S)$ of every primitive tandem repeat type can be reached by a chain of successful suffix-link walks from the endpoint of some primitive tandem repeat in Q. A *minimal p-sufficient* set Q is a p-sufficient set satisfying the condition that the removal of any tandem repeat type from Q creates a set that is no longer p-sufficient.

Note that if $\alpha\alpha = wa$ is in a minimal p-sufficient set Q, then either wa is not the right-rotation of another tandem repeat type aw in S, or all rotations of $\alpha\alpha$ are tandem repeats in S, but none of these is in Q except for $\alpha\alpha$ itself. (Due to the second possibility, a minimal p-sufficient set of S is not necessarily unique.)

7.1 The First Phase

In the first phase of the algorithm, we identify a minimal p-sufficient set of tandem repeats; we use that set in the second phase to find the endpoints in the suffix tree of all the distinct primitive tandem array types.

A minimal p-sufficient set can easily be found in linear time, using the set of all primitive tandem repeats (whose endpoints in $T(S)$ are found as described in Section 6.2). Let W denote the set of endpoints in $T(S)$ of all the primitive tandem repeat types. Algorithm 4 will mark all the points in W, using marks of M, C and R, whose meanings will be explained later. Initially, all endpoints in W are marked M. The following filtering procedure then finds a minimal p-sufficient set.

Algorithm 4
In any order, consider each endpoint (denoted v, but not necessarily a node) in W exactly once, and do the following.

1. If the endpoint v is marked R, then end.
2. Set v' to v.
3. Perform a suffix-link walk from v'.
4. If the walk is unsuccessful then end.
5. If the walk ends at point v, then mark v with C, and end.
6. Otherwise, set v' to the endpoint reached in the walk; mark v' with R and go to step 3.

At the end of the algorithm, the tandem repeats marked M or C are placed in a set called MC. We claim that set MC forms a minimal p-sufficient set. The distinction between tandem repeats marked M and those marked C will be explained in the next section. Minimality of MC is easy to establish. For primitivity, note that a successful walk from tandem repeat aw ends at the endpoint of a tandem repeat wa, and that aw is a primitive tandem repeat if and only if wa is also (this is proved in [SG98]). We leave the full proof that MC is a p-sufficient set to the reader.

We also claim that the running time of Algorithm 4 is $O(n)$. That can be proven by an extension of Lemma 5, and we leave the details to the reader. Also, in the same time bound, the algorithm can mark the endpoint in $T(S)$ where each marked tandem repeat is found. We assume that this is done in Phase I.

7.2 The Second Phase

To begin to describe Phase II, we first state the following key fact, whose proof is immediate.

Lemma 6. Let β be a right rotation of string α. If α^i is a tandem array somewhere in S, for $i > 2$, then the β^{i-1} must also be a tandem array in S. Reversing roles, if there is no occurrence of β^i in S, then there cannot be an occurrence of α^{i+1} in S.

Lemma 6 tightly connects the length of the longest tandem array of α's with the length of the longest tandem array of any rotation of α: those lengths can differ by at most one. In Phase II, the algorithm successively focuses on each minimal p-sufficient tandem repeat $\alpha\alpha$ in MC, trying to

find the length of the longest tandem array of α's, along with the length of the longest tandem array of each rotation of α. In particular, for each i such that α^i is in S, and β^i is also in S for each right rotation β of α, the algorithm determines whether α^{i+1} is in S, and then uses a modified version of a suffix-link walk to determine whether β^{i+1} is in S, for every rotation β of α. By Lemma 6, this process ends at a value of i where some β^{i+1} (including possibly α^{i+1}) is found to be missing from S.

We now develop this in more detail. Assume $\alpha = a\gamma$, and we know the endpoint in $T(S)$ of string $\alpha^i\delta$, where $i \geq 2$, $\alpha \in \Sigma^+$ and $\alpha^i\delta$ is the longest prefix of α^{i+1} that occurs anywhere in S. Hence $\delta = \alpha$ if and only if α^{i+1} occurs in S. For each right rotation β of α, we would like to find the endpoint in $T(S)$ of the longest prefix of β^{i+1} that occurs in S. We describe how this is done for the first rotation of α, i.e., for $\beta = \gamma a$.

We start by executing a standard suffix-link walk performed from the endpoint of $\alpha^i\delta$, arriving at the endpoint of string $(\gamma a)^{i-1}\gamma\delta$, which is guaranteed to be in $T(S)$. Note that during this suffix-link walk, we use the standard skip/count trick when appropriate (see a more detailed discussion in Section 5). Next, the walk continues down the tree (explicitly comparing characters along the path) to find the endpoint in $T(S)$ of the longest prefix of $(\gamma a)^{i+1}$ that occurs in S. We call that point v, although it may not be a node in $T(S)$. We call this walk a *generalized suffix-link walk*. Note that if it is started at the endpoint of a tandem array, and if the right-rotation of that tandem array also occurs in the suffix tree, then the generalized suffix-link walk is identical to the standard suffix-link walk.

To find the longest prefix of β^{i+1} that occurs in S, for β defined by two right rotation steps of α, simply start from v and execute another generalized suffix-link walk. Continuing in this for exactly $|\alpha|$ generalized suffix-link walks, the algorithm finds for each rotation β of α, the endpoint in $T(S)$ of the longest prefix of β^{i+1} that occurs in S. Moreover, those $|\alpha|$ walks end exactly where they began, i.e, at the endpoint in $T(S)$ of $\alpha^i\delta$. We call such a series of generalized suffix-link walks a suffix-link *cycle*.

Lemma 7. Any suffix-link cycle, started at the endpoint of $\alpha^i\delta$, takes $O(|\alpha|)$ time.

Proof. First, the cycle traverses exactly $|\alpha|$ suffix links, and those traversals take $O(|\alpha|)$ time. So we only have to show that the number of node skips

(using the standard skip/count trick), and the number of explicit character comparisons in all the down-traversals is bounded by $O(|\alpha|)$.

To count the number of node skips, we consider how the node-depths change as points are traversed during a cycle. Recall that the node-depth of a point v in $T(S)$ is the number of nodes on the path from the root to v. Each suffix-link traversal decreases the node-depth by at most one, and each node skip increases the node-depth by exactly one. Since the node depth at the end of the cycle is exactly the same as it is at the start, the number of node skips cannot exceed the number of suffix-link traversals, which is exactly $|\alpha|$.

Now we consider the number of explicit character comparisons done during a cycle. Each generalized suffix-link walk ends the first time a mismatch occurs, hence there are at most $|\alpha|$ mismatches during a cycle. To bound the number of matches, recall we are finding for each rotation β of α, the longest prefix of β^{i+1} that occurs in S. We bound the number of matches that occur during a cycle by considering pl which is defined as the length of that prefix minus $i \times |\alpha|$. The key observation is that if $pl = k$ after some generalized suffix-link walk, then $pl \geq k - 1$ after the next generalized suffix-link walk. For example, if $(a\gamma)^i a\phi$ is the longest prefix of $(a\gamma)^{i+1}$ in S, then $(\gamma a)^i \phi$ is a prefix of $(\gamma a)^{i+1}$ and it is also in S. That means that if there are m character comparisons that are matches during a generalized suffix-link walk, then pl after that walk is exactly $m - 1$ larger than before the walk. Since pl must be bounded by $|\alpha|$, the total number of matches that occur during a cycle is bounded by $2|\alpha|$. □

7.3 Algorithm 5

We now present the algorithm that decorates the suffix tree with the endpositions of all primitive tandem arrays, given the endpositions of all the tandem repeats from a minimal p-sufficient set MC.

The endpoints in $T(S)$ of all the primitive tandem repeats have already been located and marked in Phase I. Moreover, if $\alpha\alpha$ is a tandem repeat in MC and is marked M, then there is some right rotation β' of α such that $\beta'\beta'$ is not in S. Hence by Lemma 6 there is no right rotation β of α (including α) such that β^3 is in S. So if β^i for $i > 2$ is a tandem array in S and β is a rotation of α, where $\alpha\alpha$ is in MC, then $\alpha\alpha$ must be labeled C. The following algorithm restricts attention to the C labeled tandem repeats, and finds the endpoints in $T(S)$ of every primitive tandem array β^i for $i > 2$.

Algorithm 5

For each endpoint of a tandem repeat $\alpha\alpha$ in MC marked C, set i to 1, and do the following:

Repeat

Set i to $i+1$.

Walk in $T(S)$ from the end of α^i to the end of the longest prefix of α^{i+1} that occurs in $T(S)$. Call that point v.

Perform a suffix-link cycle starting at v to locate in $T(S)$ the endpoints of every tandem array β^{i+1} that occurs is S, where β is a right rotation of α.

Until there is a β (possibly α) such that β^{i+1} is not in S.

7.3.1 Analysis of Algorithm 5

The endpoint of every tandem array β^i for $i > 2$ is located at least once by Algorithm 5 because MC is p-sufficient, and it is located at most once because MC is minimal. In more detail, if $\alpha\alpha$ and $\alpha'\alpha'$ are two C marked tandem repeats in MC, then β is a rotation of at most one of the strings α or α'. If that were not true, then MC would not be minimal. Hence the endpoint of every tandem array β^i for $i > 2$ is located exactly once by Algorithm 5. Because of this non-redundancy, the time analysis of Algorithm 5 can concentrate separately on the time used for each execution of Algorithm 5, i.e., for each C-labeled tandem repeat in MC.

Time complexity

We have already established that Phase I takes $O(n)$ time, so we analyze the time used for Algorithm 5. For a fixed C-labeled tandem repeat $\alpha\alpha$, each iteration of the steps inside the Repeat statement involves at most $|\alpha|$ individual character comparisons, followed by the traversal of a suffix-link cycle. So, using Lemma 7, the time for an iteration is $O(|\alpha|)$.

By construction, at the start of each iteration of the Repeat statement (where i is set say to k), it is known that β^k is in S for each of the $|\alpha|$ rotations β of α. Hence, when i is set to k, the time to execute the statements inside the Repeat statement can be charged to the $|\alpha|$ tandem arrays β^k. It follows that the time for Algorithm 5 is proportional to z, the number of primitive tandem arrays in S, and the total time for both phases is $O(n+z)$.

7.4 Final extension

Note that the above algorithm starts with the suffix tree decorated with the endpoints of all the *primitive* tandem repeats occurring in S, and finds the endpoints of all the primitive tandem arrays occurring in S. Similarly, if one starts the algorithm with *all* the tandem repeat endpoints marked, one gets also the endpoints of all the tandem arrays marked, even though in most cases we only wish to locate the primitive tandem arrays.

8 Conclusion

Linear-time decoration of the suffix tree of S with the endpoints of the $O(n)$ tandem repeat types provides a compact representation of all the tandem repeat occurrences and tandem repeat types in a string. Many problems concerning tandem repeats can then be easily solved using this decorated suffix tree, resulting in algorithms which are as fast, as space efficient (and simpler) or faster, or more space efficient, or both (but perhaps not simpler) than previously proposed algorithms.

Implementations of the algorithms discussed in this paper (and other algorithms concerning repeats) can be found at:

`http://www.cs.ucdavis.edu/~gusfield/strmat.html`

References

[AP83] A. Apostolico and F. P. Preparata. Optimal off-line detection of repetitions in a string. *Theor. Comput. Sci.*, 22:297–315, 1983.

[CR94] M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, New York, NY, 1994.

[Cro81] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Inf. Process. Lett.*, 12(5):244–250, 1981.

[Cro83] M. Crochemore. Recherche linéaire d'un carré dans un mot. *C. R. Acad. Sci., Paris*, 296:781–784, 1983.

[Cro86] M. Crochemore. Tranducers and repetitions. *Theor. Comput. Sci.*, 45:63–86, 1986.

[Far97] M. Farach. Optimal suffix tree construction with large alphabets. In *Proc. 38th Annu. Symp. Found. Comput. Sci., FOCS 97*, pages 137–143, New York, NY, 1997. IEEE Press.

[FS98] A. S. Fraenkel and J. Simpson. How many squares can a string contain? *J. Comb. Theory Ser. A*, 82:112–120, 1998.

[Gus97] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, NY, 1997.

[KK98] R. Kolpakov and G. Kucherov. Maximal repetitions in words or how to find all squares in linear time. Technical Report 98-R-227, LORIA, 1998.

[Kos94] S. R. Kosaraju. Computation of squares in a string. In M. Crochemore and D. Gusfield, editors, *Combinatorial Pattern Matching: 5th Annual Symposium, CPM 94. Asilomar, California, June 1994. Proceedings*, number 807 in Lecture Notes in Computer Science, pages 146–150, Berlin, 1994. Springer Verlag.

[LS93] G. M. Landau and J. P. Schmidt. An algorithm for approximate tandem repeats. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching: 4th Annual Symposium, CPM 93. Padova, Italy, June 1993. Proceedings*, number 684 in Lecture Notes in Computer Science, pages 120–133, Berlin, 1993. Springer Verlag.

[LZ76] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Trans. Inf. Theory.*, 22(1):75–81, 1976.

[Mai89] M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25:145–153, 1989.

[McC76] E. M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976.

[ML84] M. G. Main and R. J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *J. Algorithms*, 5:422–432, 1984.

[ML85] M. G. Main and R. J. Lorentz. Linear time recognition of squarefree strings. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume F12 of *NATO ASI Series*, pages 271–278. Springer Verlag, Berlin, 1985.

[RPE81] M. Rodeh, V. R. Pratt, and S. Even. Linear algorithm for data compression via string matching. *J. ACM*, 28(1):16–24, 1981.

[SG98] J. Stoye and D. Gusfield. Simple and flexible detection of contiguous repeats using a suffix tree. In M. Farach, editor, *Combinatorial Pattern Matching: 9th Annual Symposium, CPM 98. Piscataway, New Jersey, USA, July 1998. Proceedings*, number 1448 in Lecture Notes in Computer Science, pages 140–152, Berlin, 1998. Springer Verlag.

[Ukk95] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995.

[Wei73] P. Weiner. Linear pattern matching algorithms. In *IEEE 14th Annual Symposium on Switching and Automata Theory*, pages 1–11. IEEE Press, 1973.