Of Periods, Quasiperiods, Repetitions and Covers

Alberto Apostolico[1,2*] and Dany Breslauer[2]

[1] Department of Computer Science, Purdue University,
West Lafayette, IN 47907, USA
[2] Dipartimento di Elettronica e Informatica, Università di Padova, Padova, Italy.
[3] Zeta Information Systems, New York, USA.

Abstract. *Quasiperiodic* strings were defined by Apostolico and Ehrenfeucht [3], as *strings which are entirly covered by occurrences of another (shorter) string.* This paper surveys a handful of results on the structure and detection of quasiperiodic strings and on related string covers, attempting to simplify and present in a uniform manner the algorithms being surveyed.

1 Introduction

Periodicities and other regularities in strings arise in various disciplines such as combinatorics, automata and formal language theory, data compression, stochastic process theory, symbolic dynamics, system theory and molecular biology. In the Summer of 1990, A. Ehrenfeucht suggested that some repetitive structures defying the classical characterizations of periods and repetitions could be captured by resort to a germane notion of "quasiperiod". In their paper "Efficient Detection of Quasiperiodicities in Strings" [3] Apostolico and Ehrenfeucht defined *quasiperiodic* strings as *strings which are entirely covered by occurrences of another (shorter) string.* They also gave an $O(n \log^2 n)$ time algorithm to find all maximal quasiperiodic substrings within a given string. Apostolico, Farach and Iliopoulos [4] gave an $O(n)$ time algorithm that finds the quasiperiod of a given string, namely the *shortest string* that covers the string in question. This algorithm was subsequently simplified and improved by Breslauer [9] who gave an $O(n)$ time on-line algorithm, and parallelized by Breslauer [10] and Iliopoulos and Park [19], the latter giving an optimal-speedup $O(\log \log n)$ time parallel CRCW-PRAM algorithm. Moore and Smyth [24] gave an $O(n)$ time algorithm that finds *all strings* that cover a given string. These developments eventually led to the study by Iliopoulos, Moore and Park [18] and by Ben-Amram et al. [5] of covers which are not necessarily aligned with the ends of the string being

* Partially supported by NSF Grant CCR-92-01078, by NATO Grant CRG 900293, by British Engineering and Physical Sciences Research Council grant GR/L19362, by the National Research Council of Italy, and by the ESPRIT III Basic Research Programme of the EC under contract No. 9072 (Project GEPPCOM).

covered, but are rather allowed to overflow on either side. The sequential algorithm for this problem takes $O(n \log n)$ time [18] and the parallel counterpart [5] achieves an optimal speedup taking $O(\log n)$ time, but using superlinear space.

This paper surveys the above mentioned articles, attempting to put the different results in a unified framework, and to simplify the algorithms. The main emphasis is put on some of the sequential algorithms while the parallel counterparts are sketched in lesser detail.

2 Preliminaries

We start by recalling the basic definitions and properties of strings that will be used throughout the paper. Lothaire's book [21] provides an excellent overview of additional periodicity properties of strings.

2.1 Periods and Repetitions

Given a string $w = w_1 w_2 \cdots w_n$, we denote its length by $|w| = n$. The individual symbols w_i are assumed to be taken from some underlying alphabet Σ. We write $w_{[i...j]}$ to specify the substring $w_i w_{i+1} \cdots w_j$, for $i \leq j$, and denote by ϵ the empty string.

The string w is said to have *a period of length* π, if $w_i = w_{i+\pi}$, for all feasible values of i. Clearly, by the definition above, $\pi = 0$ is a period length of w and any $\pi \geq |w|$ is also a period length. In additions, $\pi < 0$ is a period length if and only if $-\pi$ is a period length as well. We shall restrict our attention, therefore, only to period lengths π of w, such that $0 \leq \pi \leq |w|$. The integers 0 and $|w|$ are always period lengths of w, and are called the *trivial period lengths*; any integer in between may or may not be a period length depending on the structure of w.

A non-empty string u, $|u| \leq |w|$, will be called *a period* of w if w is a substring of u^k, for some integer $k \geq 1$. Clearly, if u is a period of w, then its length $|u|$ is a period length of w, since $|u|$ is a period length of u^k. Moreover, if $u = xy$, then any *rotation* yx of u is also a period of w since $(yx)^{k+1} = y(xy)^k x = yu^k x$ contains w as a substring. Note that this terminology is slightly different from the standard definition of a period, in that the latter requires that u is also a prefix of w. In this paper, a period u of w that is also a prefix of w is called a *left aligned period.* Clearly, given any period length $\pi > 0$ of w, the prefix $w_{[1...\pi]}$ is a left aligned period of w.

A period u is in fact a *regular cover* of w, where occurrences of u appear in w spaced exactly $|u|$ positions apart (other occurrences are also allowed) and the occurrences on the sides can overflow. Given any period u of w, consider the rotation $\hat{u}$ of u such that $\hat{u}$ is also a prefix of w (in other words, $\hat{u}$ is the rotation of u that is a left aligned period of w). If $w = \hat{u}^k$ for some integer k, namely if the regular cover of w by u is also right aligned, then w is said to have an *aligned regular cover* u. If w has no proper aligned regular covers (w itself is always a cover) then w is said to be *primitive.*

The following Theorem due to Fine and Wilf [16] is of fundamental importance in the study of periodic strings:

Theorem 1. *If a string w has two period of lengths p and q, and $|w| \geq p + q - \gcd(p, q)$, then w has also a period of length $\gcd(p, q)$.*

The shortest non-zero period length of w will be called *the* period length of w and denoted $\pi(w)$. A string w such that $|w| \geq 2\pi(w)$ is said to be *periodic*. By the theorem above, in a periodic string w, all periods lengths that are smaller than $|w|/2$, must be multiples of *the period* length $\pi(w)$.

2.2 General Covers

One may generalize the notion of a period u that covers w with regular occurrences that are $|u|$ positions apart in w, to covers where the occurrences of u in w are not required to be uniformly spaced, and are allowed, in addition, to overflow on either side. For example, the string $w =$ '*aabaabab*' may be covered by occurrences of $u =$ '*aba*', but the positions of these occurrences in w are not regular and in fact *aba* is not a period of w. This type of covers were called *general covers* in [18] where a covering string such as our u above is also termed *a seed* of w.

2.3 Aligned Covers

Some notable families of covers result by considering covering strings u for w that are not necessarily regularly spaced but are aligned on both sides of w and are not allowed to overflow. Such strings u are said to be aligned covers of w. Given the similarity between non-regular covers and regular covers (periods), aligned covers u of w were named *quasiperiods* of w by Apostolico and Ehrenfeucht [3]. In addition, strings that do not have any non-trivial (shorter) aligned covers were called *superprimitive* and strings that have shorter aligned covers were termed *quasiperiodic*. Observe that any periodic string is also quasiperiodic, but not every quasiperiodic string is periodic. Most of the treatment of the present paper is confined to aligned covers, leaving general covers, that are handled differently, to future extensions.

2.4 Borders

We say that a non-empty string z is a *border* of a string w if w begins and ends with an occurrence of z. Namely, $w = zu$ and $w = vz$ for some possibly empty strings u and v. Clearly, a string is always a border of itself. This border is called the *trivial* border.

We describe next few facts about periods, borders, and aligned covers.

Fact 2. *A string w has a period of length π, such that $\pi < |w|$, if and only if it has a non-trivial border of length $|w| - \pi$.*

Proof. Immediate from the definitions of a border and a period.

Fact 3. *If a string z aligned-covers a string w then z is a border of w.*

Proof. Since the first symbol of w must be covered by z, the string w must start with an occurrence of z. Since the last symbol of w must also be covered by z, the string w must also end with an occurrence of z. That is, z is a border of w.

Note that by this last Fact any cover of a string w can be represented by a single integer that is the length of the border of w.

Fact 4. *If a string z covers a string w, then z covers also any possible border v of w such that $|v| \geq |z|$.*

Proof. Given any prefix of w, it is covered by z except possibly at most the last $|z| - 1$ symbols of the prefix. Similarly, given any suffix of w, it is covered by z except possibly at most the first $|z| - 1$ symbols of the suffix. Since v is a border of w, it is both a prefix and a suffix, and it must be covered by z.

Fact 5. *Every string has a unique quasiperiod.*

Proof. Assume that a string w is covered by two strings u and v, and let w.l.o.g. $|u| \leq |v|$. By Fact 3 v is a border of w. By Fact 4 u covers w. Since $u \neq v$, then v is quasiperiodic.

Fact 6. *If a string w has a border z, such that $2|z| \geq |w|$, then z covers w.*

Proof. z covers the first half of w since it is a prefix of w and the last half of w since it is also a suffix. Therefore, all symbols of w are covered by z.

3 Finding Aligned Covers

The essentials of all the algorithms that find quasiperiods are extremely similar and were described in the paper by Apostolico, Iliopolous and Farach [4]. We outline the ideas first and then present one algorithm in details.

1. **Candidates.** By Fact 3 only borders of w are candidates to be aligned covers of w.
2. **Elimination.** Given two borders u and v of w, such that $|u| < |v|$, one of them can be eliminated as being the quasiperiod of w by Fact 4, since if u covers v then v cannot be the quasiperiod of w and if u does not cover v, then surely u cannot cover w.

3.1 Computing Borders and Periods

In order to utilize the basic ideas mentioned above, the algorithms for finding covers of a string need first to find the borders of that string. There are well established algorithms for the more or less explicit computation of borders, and they will only be mentioned here without further details.

The classical Knuth-Morris-Pratt [20] string searching algorithm computes in its pattern processing step the so called *failure function* of the pattern string that is essentially a table of border lengths of every prefix of the pattern. As seen above, this is computationally equivalent to computing the period lengths of all prefixes of the pattern. In fact, we can interpret the Knuth, Morris and Pratt [20] pattern preprocessing algorithm as an on-line algorithm that finds the period length of each prefix of a string $w_{[1...n]}$ while the string is being read in one symbol at a time. The algorithm takes $O(n)$ time and uses linear auxiliary space. The number of symbol comparisons performed by the algorithm is less than $2n$.

The parallel string cover algorithm uses the parallel CRCW-PRAM algorithm of Breslauer and Galil [2, 12] that finds all periods of a string of length n in $O(\log\log n)$ time using $n/\log\log n$ processors.

3.2 Sequential Algorithm for Quasiperiods

The first algorithm for finding the quasiperiod of a string is due to Apostolico, Farach and Iliopoulos [4]. The algorithm used the ideas above in a recursive paradigm resulting in $O(n)$ time. Breslauer [9] devised an algorithm that works *on-line*, i.e., it finds the quasiperiod of all pattern prefixes as these prefixes are are produced consecutively, one symbol at a time. This algorithm, which is outlined next, also requires fewer symbol comparisons.

The idea in the algorithm is to maintain on-line, as soon that the input prefix $w_{[1...k]}$ is given, the quasiperiod of this prefix. When a longer prefix $w_{[1...k]}$ is given, its quasiperiod is computed by observing that is must either be $w_{[1...k]}$ itself (whence $w_{[1...k]}$ is superprimitive) or it must be the quasiperiod of the longest non-trivial border of $w_{[1...k]}$ (in which case $w_{[1...k]}$ is quasiperiodic).

The algorithm scans the input string $w_{[1...n]}$ one symbol at a time. It maintains two arrays: $Quasi[i]$ and $Reach[i]$. The $Quasi[i]$ array stores the quasiperiod of any prefix $w_{[1...i]}$, for $1 \leq i \leq k$. The $Reach[i]$ array is used only for superprimitive prefixes of w and it stores, for every such prefix $w_{[1...i]}$, the longest prefix of $w_{[1...k]}$ that is covered by $w_{[1...i]}$. Note that the prefix $w_{[1...i]}$ is superprimitive if and only if $Quasi[i] = i$. When the algorithm proceeds to the next symbol $w_{[k]}$, it has to compute the quasiperiod of $w_{[1...k]}$ and store it in $Quasi[k]$.

As soon that the next input symbol $w_{[k]}$ is reached, the algorithm calls the Knuth-Morris-Pratt algorithm to find the period length π_k of the prefix $w_{[1...k]}$. The only comparisons of input symbols are performed in these calls to the Knuth-Morris-Pratt algorithm. Once the period of the prefix $w_{[1...k]}$ is given, the algorithm can proceed to compute the values of $Quasi[k]$ and $Reach[Quasi[k]]$ based only on the values of π_k, $Quasi[1 \ldots k-1]$ and $Reach[1 \ldots k-1]$. There are two cases:

1. If $\pi_k = k$, then by Fact 2 the prefix $w_{[1...k]}$ has no non-trivial border. By Fact 3 any string that covers $w_{[1...k]}$ must also be a border. Thus, the prefix $w_{[1...k]}$ is covered only by itself and therefore it is superprimitive. In this case we define $Quasi[k] = k$ and $Reach[k] = k$.

- The $Quasi[i]$ array stores the quasiperiod of any prefix $w_{[1...i]}$.
- The $Reach[i]$ array stores only for superprimitive prefixes $w_{[1...i]}$ the longest
- prefix of $w_{[1...k]}$, that is covered $w_{[1...i]}$.

```
k = 1
while k ≤ n do
    Compute the period π_k of w_[1...k] using the Knuth-Morris-Pratt
      pattern preprocessing algorithm.
    - If the prefix w_[1...k] has a non-trivial border check if the quasiperiod
    -   of that border covers the whole prefix.
    if π_k < k and Reach[Quasi[k − π_k]] ≥ k − Quasi[k − π_k] then
        - If the quasiperiod of the border w_[1...k−π_k] covers the whole
        -   prefix w_[1...k], then it is also the quasiperiod of w_[1...k].
        Quasi[k] = Quasi[k − π_k]
        Reach[Quasi[k]] = k
    else
        - If the prefix w_[1...k] does not have any non-trivial border
        -   or if the quasiperiod of the border does not cover the whole
        -   prefix w_[1...k], then it is superprimitive.
        Quasi[k] = k
        Reach[k] = k
    end
    k = k + 1
end
```

Fig. 1. The quasiperiodicity algorithm.

2. If $\pi_k < k$, then by Fact 2 the prefix $w_{[1...k]}$ has a border of length $k - \pi_k$. Since π_k is the shortest period of $w_{[1...k]}$, the border of length $k - \pi_k$ is the longest non-trivial border of the prefix $w_{[1...k]}$.

   If the prefix $w_{[1...k]}$ is quasiperiodic, then by Fact 4 the border $w_{[1...k-\pi_k]}$ must also be covered by the same quasiperiod. The algorithm checks if the quasiperiod of the border $w_{[1...k-\pi_k]}$ can cover the whole prefix $w_{[1...k]}$. If a cover of the prefix $w_{[1...k]}$ by the quasiperiod of $w_{[1...k-\pi_k]}$ is given, then a cover of a shorter prefix of $w_{[1...k]}$ is obtained by removing the last occurrence of that quasiperiod. This means that the quasiperiod of $w_{[1...k-\pi_k]}$ covers a prefix of $w_{[1...k]}$ that is long enough that with one more occurrence of that quasiperiod, the whole prefix $w_{[1...k]}$ is covered.

   Thus, all the algorithm has to do is to check if the quasiperiod of the border $w_{[1...k-\pi_k]}$ can cover a prefix of $w_{[1...k]}$ that is long enough. That is, if $Reach[Quasi[k - \pi_k]] \geq k - Quasi[\pi_k]$, then that quasiperiod covers $w_{[1...k]}$ and we define $Quasi[k] = Quasi[k - \pi_k]$ and update $Reach[Quasi[k]] = k$. Otherwise, the prefix $w_{[1...k]}$ is superprimitive and we define $Quasi[k] = k$ and $Reach[k] = k$.

Theorem 7. *The algorithm that is described above and in Figure 1 takes $O(n)$ time and uses linear auxiliary space. The number of comparisons of input symbols is at most $2n$.*

3.3 Parallel Algorithm for Quasiperiods

The parallel algorithms described in this paper are for the concurrent-read concurrent-write parallel random access machine model. We use the weakest version of this model called the *common CRCW-PRAM*. In this model many processors have access to a shared memory. Concurrent read and write operations are allowed at all memory locations. If several processors attempt to write simultaneously to the same memory location, it is assumed that they write the same value.

One of the major issues in the design of PRAM algorithms is the assignment of processors to their tasks. The problem is easier when the input is rigidly allocated, e.g., on an array, as is the case with strings. In this case, we can effectively resort to a powerful general principle which ignores the issue of processor assignment.

Theorem 8. *[Brent [8]] Any synchronous parallel algorithm of time t that consists of a total of x elementary operations can be implemented on p processors in $\lceil x/p \rceil + t$ time.*

The optimal speedup parallel quasiperiodicity algorithm we describe next is a variation on the algorithm given by Iliopolous and Park [19] improving on a similar non-optimal algorithm by Breslauer [10], utilizing a newly discovered parallel string searching algorithm. It uses two other parallel algorithms that are out of the scope of this paper.

1. The parallel string searching algorithm of Cole et al. [14] that finds all occurrences of a pattern string of length m in a text string of length n in constant time using n processors after a pattern preprocessing step that requires $O(\log\log m)$ time using $m/\log\log m$ processors.
   By a lower bound of Breslauer and Galil [11], this algorithm is the fastest possible optimal parallel string matching algorithm on a general alphabet, where input symbols are accessed only by pairwise comparisons. Breslauer [10] shows that the same lower bound also applies to finding the quasiperiod of a string.
   This algorithm is used in conjunction with the following algorithm to test if a given string z covers another string u.
2. The algorithm of Fich, Ragde and Wigderson [15] to compute the minimum of n integers between 1 and n in constant time using n processors.

The parallel algorithm starts by computing all the borders of the input string $w_{[1\ldots n]}$ in $O(\log\log n)$ time and $n/\log\log n$ processors using the algorithm by Breslauer and Galil [12]. The borders are partitioned according to their length

into at most $\log n$ groups $[2^i \cdots 2^{i+1} - 1]$. By Fact 6 and the observations above about elimination of quasiperiodicity candidates, only the shortest border in each group is a candidate to be the quasiperiod of w. This provides an easy elimination of all but at most $\log n$ candidates, which can be carried out in constant time by n processors using the integer minima algorithm that is mentioned above, in each interval separately, but in parallel.

The main step that underlies the rest of the algorithm is a procedure that tests, given two strings u and v, if u covers v. This is carried out by using the constant time string searching algorithm to find all occurrences of u in v using $|v|$ processors, provided that u has been already preprocessed, and then applying the integer minima (maxima) algorithm mentioned above in each interval of consecutive $|u|$ positions of v to check that the occurrences of u are not spaced too far apart in v. Clearly, all the $O(\log n)$ candidates for the quasiperiod can be preprocessed for the string searching algorithm simultaneously in $O(\log \log n)$ and using n processors. This preprocessing can be later used for searching the same preprocessed pattern on multiple occasions.

The algorithm first eliminates all but at most one candidate of those with length smaller than or equal to $n/\log n$. This is done by picking the longest candidate shorter than $n/\log n$ and testing simultaneously in constant time using n-processors if each of the $O(\log n)$ shorter candidates covers that longest candidate. The shortest candidate to cover the longest candidate with length at most $n/\log n$ is the only remaining candidate among those with lengths at most $n/\log n$.

After this elimination step, we are left with at most $1 + \log \log n$ quasiperiod candidates. Specifically, there will be at most one candidate shorter than $n/\log n$ and at most $\log \log n$ longer candidates. The algorithm proceeds by picking in each step the shortest two remaining candidates and eliminating one of them using the cover test above. This takes constant time for each step and $O(\log \log n)$ time in total. The number of operations used sums up to be $O(n)$ since it is bounded by the sum of lengths of all candidates.

The algorithm therefore is composed of several steps taking $O(\log \log n)$ time, using $n/\log \log n$ processors or steps taking $O(\log \log n)$ time and making $O(n)$ operations. The steps can be combined and slowed down to get the following result.

Theorem 9. *The quasiperiod of a string can be found in $O(\log \log n)$ time using $n/\log \log n$ processors and linear space.*

4 Finding All Covers

Moore and Smyth [24] gave an $O(n)$ time algorithm that finds *all covers* of a given string. At the heart of the algorithm is a procedure that finds the longest proper aligned cover of a string in $O(n)$ time, by looking, essentially, for the longest cover of the string which is aligned on its left but may overflows on its right.

We do not give the details of that algorithm in this short survey. We remark, however, that one can also easily find all covers of a string in $O(n \log n)$ time sequentially, and in $O(\log \log n)$ time with optimal speedup in parallel, using the ideas presented in the sequential and parallel algorithms above.

5 Finding Maximal Quasiperiodic Substrings

Assume that we are given a string w of length $n = |w|$. Consider a segment $u = w_{[i \ldots i+h-1]}$ of length $h = |u|$, starting at position i, and its quasiperiod z. Apostolico and Ehrenfeucht named the triple (i, z, h) *quasiperiodic span*, and say that the quasiperiodicity is *maximal* if the following two conditions are satisfied:

1. The quasiperiodic span can not be extended on either side by more occurrences of z. Namely, there is no other quasiperiodic span (i', z, h') such that $i' \leq i$ and $i' + h' \geq i + h$.
2. The quasiperiodic span can not be extended on the right by one more symbol when the covering string is also extended by the same symbol. Namely, if $a = w_{i+|u|}$, then za does not cover ua, or in other words, ua does not have the same quasiperiod as za.

Apostolico and Ehrenfeucht were interested in finding all maximal quasiperiodic substrings of a given string. Their algorithm is based on suffix trees and properties of their structures, which are given next. For a survey of other applications of suffix trees see [1, 17].

Suffix trees Suffix trees are a compressed form of *digital search trees* that are very useful in many algorithms on strings. The usual definition of a suffix tree is the following:

Let w be a string having as its last symbol a special marker '#' that does not appear anywhere else in w. The *suffix tree* of w is a rooted tree with $|w|$ leaves and $|w| - 1$ internal nodes such that:

1. Each edge is labeled with a non-empty substring of w.
2. No two sibling edges are labeled with substrings that start with the same symbol.
3. Each leaf is labeled with a distinct position of w.
4. The concatenation of the labels of the edges on the path from the root to a leaf labeled with position i yields precisely the suffix $w_{[i \ldots |w|]}$.

An example of a suffix tree is given in Figure 2. Note that a suffix tree has no internal nodes with one child. Substrings of w can be represented by their starting and ending positions and therefore a suffix tree can be stored in $O(|w|)$ space.

We shall identify the name of a suffix tree node with the concatenation of the edge labels from the root to that node. If two suffixes $w_{[i \ldots k]}$ and $w_{[j \ldots k]}$ have the same prefix, namely if $w_{[i \ldots i+l-1]} = w_{[j \ldots j+l-1]}$ and $w[i + l] \neq w[j + l]$, then the

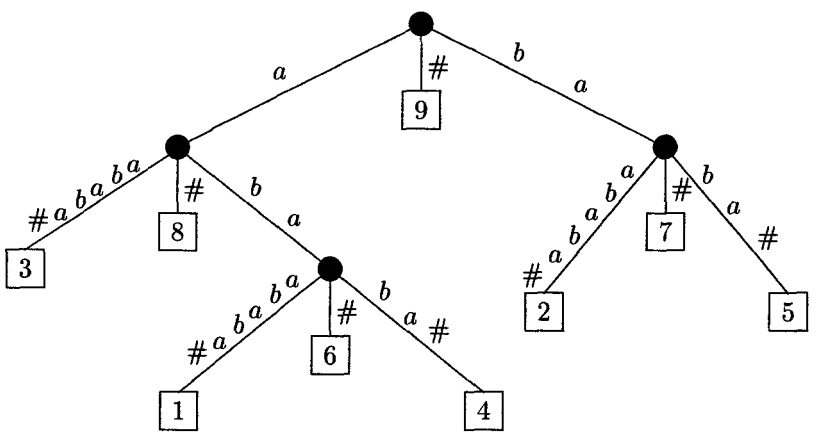

Fig. 2. The suffix tree of w = '*abaababa*#'. The edges are labeled with substrings of w and the leaves with the position in w at which the corresponding suffix starts.

paths from the root to the leaves labeled with positions i and j share an initial segment. The concatenation of the labels of the edges on this initial segment is equal to $w[i \ldots i + l - 1]$. Similarly, for any substring u of w, there must either be a unique node in w such that the concatenation of labels on the path from the root to the node is equal to uv with v being of minimal length and possibly empty. If there is such a node labeled u, it is called the proper *locus* of u and if it is labeled uv with $v \neq \epsilon$, then it is called the *extended locus* of u.

The special alphabet symbol '#' that was assumed to be the last symbol of the string $w[1..k]$ is normally appended at the end of a given string to guarantee that the suffix tree has distinct leaves that correspond to each suffix. There exist several efficient algorithms to construct suffix trees and important related structures such as inverted files and subword automata [6, 7, 13, 23, 22, 25, 26].

5.1 Suffix Trees and Maximal Quasiperiodicities

We next give some relations, that were given by Apostolico and Ehrenfeucht, between suffix trees and maximal quasiperiodicities.

Lemma 10. *Let $(i, z, |u|)$ be a maximal quasiperiodicity in w. Then, z must have a proper locus in the suffix tree of w.*

Proof. Assume on the contrary that z only has an extended locus zv and let a be the first symbol of v. By properties of the suffix tree, all occurrences of z in w must also be occurrences of za and therefore the substring $w_{[i \ldots i+|u|]}$ must be covered by za, what contradicts the maximality of $(i, z, |u|)$.

By the last Lemma, it suffices to consider the superprimitive node labels in the suffix tree as superprimitive strings that have a maximal quasiperiodicity.

Given a node labeled z in the suffix tree, one might consider the substrings of w that are maximal quasiperiodic substrings with quasiperiod z. Clearly, there must be occurrences of z that cover each of these substrings, and by the definition of the suffix tree, each occurrence of z in w will have a leaf corresponding to its starting position as a descendant of the node z. Define a *run* at node z to be a maximal sequence of positions in w where occurrences of u start and that are not more than $|z|$ positions apart from each other. A run corresponds to a maximal substring of w that is covered by u and not contained in any longer substring that is covered by z.

We say that a run *coalesces* at a node z of the suffix tree, if it is a run at z but not at any of its children. This terminology allows us to characterize the maximal quasiperiodicities precisely.

Theorem 11. *(i, z, h) is a maximal quasiperiodicity in w if and only if there is a node in the suffix tree of w labeled z and a run $\{i_1 < i_2 < \cdots < i_k\}$, $k \geq 2$, that coalesces at z, such that:*

1. *$i = i_1$ and $i_k = i + h - |z|$. Namely, $u = w_{[i_1 \ldots i_1+h-1]}$ can not be extended on either side by more occurrences of z.*
2. *There is no ancestor z' of z where the position i_1 is in the same run with the position $i_1 + |u| - |z'|$. Namely, z is not covered by a shorter string z' and therefore, is superprimitive.*

The algorithm of Apostolico and Ehrenfeucht is built around the Theorem above and it finds the maximal quasiperiodicities by maintaining the runs for each node of the suffix tree while climbing bottom-up from the leaves. Observe that as the algorithm progresses computing the runs at a node given the runs of their children, runs of children split since the length of the parent node label is shorter than the children, and other runs merge. Apostolico and Ehrenfeucht gave an algorithm that takes $O(n \log^2 n)$ time.

6 Open Questions

Some remaining open questions are the following:

1. *All Aligned Covers.* Finding all aligned covers of all prefixes of a string in $O(n)$ time; namely, the longest proper aligned cover of each quasiperiodic prefix. An optimal parallel algorithm for finding all aligned covers of a string. These are probably the easier problems in this list.
2. *Maximal Quasiperiodic Substrings.* Improving on the $O(n \log^2 n)$ algorithm that was outlined above. Designing and efficient parallel version of this algorithm.
3. *General Covers.* Improving on the $O(n \log n)$ sequential algorithm and on the exisiting parallel algorithm.

References

1. A. Apostolico. The Myriad Virtues of Subword Trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO ASI Series F*, pages 85–96. Springer-Verlag, Berlin, Germany, 1985.
2. A. Apostolico, D. Breslauer, and Z. Galil. Optimal Parallel Algorithms for Periods, Palindromes and Squares. In *Proc. 19th International Colloquium on Automata, Languages, and Programming*, number 623 in Lecture Notes in Computer Science, pages 296–307. Springer-Verlag, Berlin, Germany, 1992.
3. A. Apostolico and A. Ehrenfeucht. Efficient Detection of Quasiperiodicities in Strings. *Theoret. Comput. Sci.*, 119:247–265, 1993.
4. A. Apostolico, M. Farach, and C.S. Iliopoulos. Optimal Superprimitivity Testing for Strings. *Inform. Process. Lett.*, 39:17–20, 1991.
5. A. Ben-Amram, O. Berkman, C. Iliopolous, and K. Park. Computing the Covers of a String in Linear Tme. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 501–510, 1994.
6. A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, M.T. Chen and J. Seiferas. The Smallest Automaton Recognizing the Subwords of a Text, *Theoretical Computer Science* , 40:31–55, 1985.
7. A. Blumer, A., J. Blumer, A. Ehrenfeucht, D. Haussler, and R. McConnell. Complete Inverted Files for Efficient Text Retrieval and Analysis. *Journal of the ACM*, 34(3): 578-595 (1987).
8. R.P. Brent. Evaluation of General Arithmetic Axpressions. *J. Assoc. Comput. Mach.*, 21:201–206, 1974.
9. D. Breslauer. An On-Line String Superprimitivity Test. *Inform. Process. Lett.*, 44(6):345–347, 1992.
10. D. Breslauer. Testing String Superprimitivity in Parallel. *Inform. Process. Lett.*, 49(5):235–241, 1994.
11. D. Breslauer and Z. Galil. A Lower Bound for Parallel String Matching. *SIAM J. Comput.*, 21(5):856–862, 1992.
12. D. Breslauer and Z. Galil. Finding all Periods and Initial Palindromes of a String in Parallel. *Algorithmica*, 1995.
13. M.T. Chen and J. Seiferas. Efficient and Elegant Subword-tree Construction. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO ASI Series F*, pages 97–107. Springer-Verlag, Berlin, Germany, 1985.
14. R. Cole, M. Crochemore, Z. Galil, L. Gąsieniec, R. Hariharan, S. Muthukrishnan, K. Park, and W. Rytter. Optimally Fast Parallel Algorithms for Preprocessing and Pattern Matching in One and Two Dimensions. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 248–258, 1993.
15. F.E. Fich, R.L. Ragde, and A. Wigderson. Relations Between Concurrent-write Models of Parallel Computation. *SIAM J. Comput.*, 17(3):606–627, 1988.
16. N.J. Fine and H.S. Wilf. Uniqueness Theorems for Periodic Functions. *Proc. Amer. Math. Soc.*, 16:109–114, 1965.
17. R. Grossi and G.F. Italiano. Suffix Trees and their Applications in String Algorithms. Manuscript, 1995.
18. C.S. Iliopoulos, D.W.G. Moore, and K. Park. Covering a String. In *Proc. 4th Symp. on Combinatorial Pattern Matching*, number 684 in Lecture Notes in Computer Science, pages 54–62, Berlin, Germany, 1993. Springer-Verlag.

19. C.S. Iliopoulos and K. Park. An Optimal $O(\log \log n)$-time Algorithm for Parallel Superprimitivity Testing. *J. Korea Information Science Society*, 21(8):1400–1404, 1994.
20. D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast Pattern Matching in Strings. *SIAM J. Comput.*, 6:322–350, 1977.
21. M. Lothaire. *Combinatorics on Words.* Addison-Wesley, Reading, MA, U.S.A., 1983.
22. U. Manber and E. Myers. Suffix Arrays: a New Method for On-line String Searches. *Proceedings of the 1st Symposium on Discrete Algorithms*, 319-327, 1990.
23. E.M. McCreight. A Space Economical Suffix Tree Construction Algorithm. *J. Assoc. Comput. Mach.*, 23:262–272, 1976.
24. D. Moore and W.F. Smyth. Computing the Covers of a String in Linear Time. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 511–515, 1994.
25. E. Ukkonen. Constructin Suffix Trees On-line in Linear Time. *Proceedings of Information Processing 92* , Vol. 1, 484–492, 1992.
26. P. Weiner. Linear Pattern Matching Algorithms. In *Proc. 14th Symposium on Switching and Automata Theory*, pages 1–11, 1973.