BRICS RS-96-21 Breslauer et al.: Rotation of Periodic Strings and Short Superstrings

BRICS

Basic Research in Computer Science

Rotation of Periodic Strings and Short Superstrings

Dany Breslauer
Tao Jiang
Zhigen Jiang

BRICS Report Series **RS-96-21**

ISSN 0909-0878 **June 1996**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark

Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through WWW and anonymous FTP:

```
http://www.brics.dk/
ftp ftp.brics.dk (cd pub/BRICS)
```

Rotations of Periodic Strings and Short Superstrings

Dany Breslauer* Tao Jiang† Zhigen Jiang‡

June 10, 1996

Abstract

This paper presents two simple approximation algorithms for the shortest superstring problem, with approximation ratios $2\frac{2}{3}$ (≈ 2.67) and $2\frac{25}{42}$ (≈ 2.596), improving the best previously published $2\frac{3}{4}$ approximation. The framework of our improved algorithms is similar to that of previous algorithms in the sense that they construct a superstring by computing some optimal cycle covers on the distance graph of the given strings, and then break and merge the cycles to finally obtain a Hamiltonian path, but we make use of new bounds on the overlap between two strings. We prove that for each periodic semi-infinite string $\alpha = a_1 a_2 \cdots$ of period q, there exists an integer k, such that for *any* (finite) string s of period p which is *inequivalent* to α, the overlap between s and the *rotation* $\alpha[k] = a_k a_{k+1} \cdots$ is at most $p + \frac{1}{2}q$. Moreover, if $p \leq q$, then the overlap between s and $\alpha[k]$ is not larger than $\frac{2}{3}(p+q)$. In the previous shortest superstring algorithms $p+q$ was used as the standard bound on overlap between two strings with periods p and q.

1 Introduction

Let $S = \{s_1, \ldots, s_m\}$ be a set of strings over some alphabet Σ. A *common superstring*, or simply *superstring*, of S is a string s such that each s_i in S is

*Max-Planck-Institute für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. Part of this work was carried out while this author was staying at BRICS – Basic Research in Computer Science, Centre of the Danish National Research Foundation, Department of Computer Science, University of Aarhus, 8000 Aarhus C, Denmark. Partially supported by ESPRIT Long Term Research Program of the EU under contract #20244 (ALCOM-IT). E-mail: breslau@mpi-sb.mpg.de

†Department of Computer Science, McMaster University, Hamilton, Ontario L8S 4K1, Canada. Supported in part by NSERC Research Grant OGP0046613 and MRC/NSERC Canadian Genome Analysis and Technology (CGAT) Grant GO-12278. Part of the work was done while on research leave at University of Washington. E-mail: jiang@maccs.mcmaster.ca

‡Supported in part by MRC/NSERC CGAT Grant GO-12278. Address: Department of Electrical and Computer Engineering, McMaster University, Hamilton, Ont. L8S 4K1, Canada. E-mail: zjiang@maccs.mcmaster.ca

a substring (*i.e.* a consecutive block) of s. The shortest superstring problem is to find a superstring of the smallest possible length for any given set of strings S. The problem has applications in a wide range of areas including data compression [11, 19] and DNA sequencing [14, 15, 18, 23]. For example, in *shotgun* DNA sequencing, a long DNA molecule[1] is first cleaved into short overlapping fragments of roughly 500 bases. Each such short fragment is then sequenced and a string over the set of nucleotides $\{A, C, G, T\}$ is obtained. From hundreds or thousands of these fragments, a biochemist tries to construct a shortest superstring representing the sequence for the whole DNA molecule.

Since the problem is *NP-hard* [11] a lot of effort has been taken to find good approximation algorithms with guaranteed performance. Blum et al. [4] showed that the problem is *MAX SNP-hard* and thus does not have a polynomial time approximation scheme unless P = NP. Tarhio and Ukkonen [20] and Turner [22] gave several approximation algorithms for the shortest superstring problem and proved that their algorithms achieve $\frac{1}{2}$-approximation with respect to the *compression* measure, or the *total overlap* between adjacent strings in a superstring. This approximation ratio has been improved to $\frac{38}{63}$ by Kosaraju et al. [13]. Tarhio and Ukkonen conjectured that their GREEDY approximation algorithm, which repeatedly merges pairs of strings with the maximum overlap until only one string is left, 2-approximates also the *length* of the shortest superstring. Notice that superstrings have the minimum length if and only if they induce the maximum total overlap. Such relation, however, does not hold for approximations, and a good approximation for the length of the shortest superstring is not necessarily a good approximation for the maximum overlap in the superstring, and vice versa.

The first constant-approximation algorithm for the length of the shortest superstring was given by Blum et al. [4], who discovered a 3-approximation algorithm and proved that the GREEDY algorithm achieves 4-approximation. Their algorithms and analysis rely on the close relation between the shortest superstring problem, that was shown by Turner [22] to be reducible to the *traveling salesman* problem, and the *cycle cover* problem. The same relation was exploited in subsequent papers that continued to improve the approximation ratio, by Teng and Yao [21] (≈ 2.89), Czumaj et al. [8] (≈ 2.83), Kosaraju et al. [13] (≈ 2.79) and Armen and Stein [1, 2] (≈ 2.75). Armen and Stein [3] have also recently obtained a $2\frac{2}{3}$-approximation algorithm, independently of our work[2]. A connection between the approximation ratio and the number of examples needed to infer a string (or DNA sequence) from randomly drawn examples in the PAC learning model is given in [15, 12]. This presents an additional motivation for lowering the approximation ratio.

Here we continue this line of work, and further improve the approximation ratio to $2\frac{2}{3} \approx 2.67$ and to $2\frac{25}{42} \approx 2.596$. The improved algorithms are similar

[1] It is about 1.8 megabases long in the *Haemophilus influenzae* Rd sequencing project [9].
[2] Our algorithms and analysis are conceptually and structurally much simpler.

to the previous algorithms in the sense that they constructs a superstring by computing some optimal cycle covers on the *distance graph* of the given input strings, and then break and merge the cycles to finally obtain a Hamiltonian path representing some superstring. The key to the improvement are new bounds on the overlap between two strings. We prove that for each periodic semi-infinite string $\alpha = a_1 a_2 \cdots$ of period q, there exists an integer k, such that for *any* (finite) string s of period p which is *inequivalent* to α, the overlap between s and the *rotation* $\alpha[k] = a_k a_{k+1} \cdots$ is at most $p + \frac{1}{2}q$. Moreover, if $p \leq q$, then the overlap between s and $\alpha[k]$ is not larger than $\frac{2}{3}(p+q)$. (The equivalence of strings will be defined in Section 2.2.) These bounds are tight. Previously, the sum of the periods was taken as the standard (tight) bound on overlap between two strings.

The algorithms and their analysis are actually very simple. We have chosen to describe both approximation algorithms since they use the bounds on the overlap between strings in different ways that might give some insight into future improvements.

We recall some basic definitions and facts in Section 2. The new overlap-rotation bound is given is Section 3. Section 4 gives the generic shortest superstring algorithm, and Sections 5-6 give the improved approximation algorithms and their analysis.

2 Preliminaries

Let $S = \{s_1, \ldots, s_m\}$ be a set of strings. Without loss of generality, we assume that the set S is "substring-free" in that no string $s_i \in S$ is a substring of any other $s_j \in S$. Most of the definitions below follow [4].

For two strings s and t, let y be the longest string such that $s = xy$ and $t = yz$ for some *non-empty* strings x and z. We call $|y|$ the (amount of) *overlap* between s and t, and denote it as $ov(s,t)$. The notion can be easily extended to the case where t is a semi-infinite string (but s has to be finite). The string x is called the *prefix* of s with respect to t, and is denoted $pref(s,t)$. Finally, we call $|pref(s,t)| = |x|$ the *distance* from s to t, and denote it as $d(s,t)$.

Given a list of strings $s_{i_1}, \ldots, s_{i_r}$, we define the superstring $s = \langle s_{i_1}, \ldots, s_{i_r} \rangle$ to be

$$pref(s_{i_1}, s_{i_2}) pref(s_{i_2}, s_{i_3}) \cdots pref(s_{i_{r-1}}, s_{i_r}) s_{i_r}.$$

That is, s is the shortest string such that $s_{i_1}, \ldots, s_{i_r}$ appear *in order* in s. It is clear that each shortest superstring for S must be $\langle s_{i_1}, \ldots, s_{i_m} \rangle$ for some permutation $i_1, \ldots, i_m$ of $\{1, \ldots, m\}$. The length of a shortest superstring of S is denoted $\mathrm{opt}(S)$ and the total overlap between adjacent strings in the shortest superstring of S is denoted $\mathrm{maxov}(S)$. Notice that $\mathrm{opt}(S) = \sum_{s_i \in S} |s_i| - \mathrm{maxov}(S)$.

2.1 Distance graph and cycle covers

The concept of a *distance graph* is central to all existing approximation algorithms for shortest superstrings. Let $G_S = (V, E, w)$ be a directed graph, where the set of vertices $V = \{s_1, \ldots, s_m\}$, the set of edges $E = \{(s_i, s_j) \mid 1 \leq i \neq j \leq m\}$, and the weight function w is the distance function $d(,)$. G_S is called the distance graph of S. If we denote the cost of a minimum weight Hamiltonian cycle on G_S as $\mathrm{TSP}(G_S)$, then obviously, for any $s_i \in S$,

$$\mathrm{TSP}(G_S) \leq \mathrm{opt}(S) \leq \mathrm{TSP}(G_S) + |s_i|.$$

In other words, a minimum weight Hamiltonian cycle on G_S would be a very good approximation of a shortest superstring of S. Since TSP is NP-hard and has no good approximation algorithms, we try to work with a relaxed version of TSP, the *cycle cover* problem (also called the *assignment* problem) defined below.

Given a directed weighted graph G, a *cycle cover* is a set of (simple) cycles such that each vertex is contained in exactly one cycle. The weight of the cycle cover is the total weight of its cycles. It is well-known that a minimum weight cycle cover on any directed weighted graph G can be computed in $O(n^3)$ time using the Hungarian algorithm [17].

Let $\mathrm{CYC}(G_S)$ be the weight of a minimum weight cycle cover of G_S. Then we have $\mathrm{CYC}(G_S) \leq \mathrm{TSP}(G_S) \leq \mathrm{opt}(S)$. Unfortunately, there is no obvious upper bound on $\mathrm{opt}(S)$ in terms of $\mathrm{CYC}(G_S)$ in general. So we have to look at the particular structures and properties of strings.

2.2 Periodicity of strings and semi-infinite strings

A string x is *a factor* of a string s if $s = x^i y$ for some positive integer i and prefix y of x (y may be empty). *The factor* of a non-empty string s, denoted $\mathit{factor}(s)$, is the *shortest* factor of s and the *period* of s is denoted $\mathit{period}(s) = |\mathit{factor}(s)|$. A semi-infinite string $s = a_1 a_2 \cdots$ is said to be *periodic* if $s = xs$ for some *non-empty* string x. The shortest such x is called the factor of s. Two (periodic semi-infinite) strings s, t are *equivalent* if their factors are cyclic shifts of each other, *i.e.* if there are strings x, y such that $\mathit{factor}(s) = xy$ and $\mathit{factor}(t) = yx$. Otherwise, they are *inequivalent*. For each string s, let s^i denotes the concatenation of i s's as well as the sequence of i consecutive s's, s^∞ denote the semi-infinite string $ss\cdots$, and $s_\infty = \mathit{factor}(s)^\infty$ denote the periodic semi-infinite string that is equivalent to s and begins with s. Note that in general $s^\infty \neq s_\infty$. The next lemma is easy to prove.

Lemma 2.1 *Suppose that string x is contained in string y and y is contained in string z. If x and z are equivalent, then y is also equivalent to x and z.*

The following facts connecting a cycle in G_S and the periodicity of the strings obtained by breaking the cycle are essentially given in [4]. We rephrase and state them here as lemmas without a proof. Let $c = s_{i_1}, \ldots, s_{i_r}, s_{i_1}$ be a cycle in G_S. Without loss of generality, assume that c has the minimum weight among all cycles in G_S containing $s_{i_1}, \ldots, s_{i_r}$. Denote the weight of c as $w(c)$. Let's call $\langle s_{i_1}, \ldots, s_{i_r}\rangle$ the string obtained by breaking the cycle c at s_{i_r}.

Lemma 2.2 *The string* $\langle s_{i_1}, \ldots, s_{i_r}\rangle$ *is a substring of* $factor(\langle s_{i_1}, \ldots, s_{i_r}\rangle)s_{i_1} = \langle s_{i_1}, \ldots, s_{i_r}, s_{i_1}\rangle$.

Lemma 2.3

$$factor(\langle s_{i_1}, \ldots, s_{i_r}\rangle) = pref(s_{i_1}, s_{i_2}) \cdots pref(s_{i_{r-1}}, s_{i_r}) pref(s_{i_r}, s_{i_1}).$$

Three corollaries follow immediately:

Corollary 2.4

$$w(c) = d(s_{i_1}, s_{i_2}) + \cdots + d(s_{i_{r-1}}, s_{i_r}) + d(s_{i_r}, s_{i_1}) = period(\langle s_{i_1}, \cdots, s_{i_r}\rangle).$$

Corollary 2.5 *The strings* $\langle s_{i_1}, \ldots, s_{i_r}\rangle, \ldots, \langle s_{i_r}, s_{i_1}, \ldots, s_{i_{r-1}}\rangle$ *are all equivalent.*

Corollary 2.6 $factor(\langle s_{i_1}, \ldots, s_{i_r}, s_{i_1}\rangle) = factor(\langle s_{i_1}, \ldots, s_{i_r}\rangle)$. *That is, the string* $\langle s_{i_1}, \ldots, s_{i_r}, s_{i_1}\rangle$ *is equivalent to* $\langle s_{i_1}, \ldots, s_{i_r}\rangle$.

Note that, in general $s_{i_1}, \ldots, s_{i_r}$ are not mutually equivalent, nor are s_{i_1} and $\langle s_{i_1}, \ldots, s_{i_r}\rangle$.

Lemma 2.7 *Let* $c' = s_{j_1}, \ldots, s_{j_l}, s_{j_1}$ *be another cycle. If* $\langle s_{i_1}, \ldots, s_{i_r}\rangle$ *is equivalent to* $\langle s_{j_1}, \ldots, s_{j_l}\rangle$, *then there exists a third cycle* $\tilde{c}$ *with weight* $w(c)$ *containing all vertices in* c *and* c'.

3 The overlap-rotation lemma

Lothaire's [16] book provides an excellent overview of combinatorial properties of periodic strings. Given a string w, we say that w is *unbordered* if it has no proper prefix that is also a suffix, *i.e.* $ov(w, w) = 0$ and $factor(w) = w$. Given a non-trivial *factorization* $w = uv$, namely a partition of w with non-empty prefix u and suffix v, the *local factor* of the factorization is defined as the shortest non-empty string that is *consistent* with both sides of the factorization. That is, the shortest string that matches the prefix u aligned at its end and also matches the suffix v aligned at its start. A non-trivial factorization $w = uv$ is called a *critical factorization* if its local factor is of the same length as $period(w)$. See Figure 1 for an example. We are now ready to state the so called *Critical Factorization Theorem.*

(a)	(b)	(c)
a | b a a a b a	a b | a a a b a	a b a | a a b a
b a b a	**a a a b a a a b**	**a a**

Figure 1: The local factors of the first three non-trivial factorizations of '*abaaaba*'. Note that in some cases the local factor can overflow to either side; this happens when the local factor is longer than the factorization prefix or suffix. The factorization **(b)** is a critical factorization.

Theorem 3.1 (Cesari and Vincent [5, 16]) *Given any $period(w) - 1$ consecutive non-trivial factorizations of a string w, at least one is a critical factorization.*

The notion of a critical factorization and Critical Factorization Theorem applies both to finite and infinite strings. The following lemma will be useful.

Lemma 3.2 *Let w be an unbordered string with the critical factorization $w = uv$. Then,*

1. *the rotation $w' = vu$ of w is also unbordered; and*
2. *vu is a critical factorization of w'.*

Proof: To see that w' is unbordered, assume on the contrary that there is a string x that is a proper prefix and suffix of w'. But then, x is consistent with both sides of the critical factorization uv of w, contradicting the definition.

To prove that $w' = vu$ is a critical factorization, assume on the contrary that there is a string x that is consistent with both sides of the factorization vu and that $|x| < |w|$. Clearly, since $w = uv$ is unbordered, $|x| > |v|$ or $|x| > |u|$. Assume without loss of generality that $|v| \leq |u|$ and therefore, $|v| < |x|$. Let $x = bv$. If $|x| \leq |u|$, then letting $u = xu' = bvu'$, we get contradiction since vu' is consistent with both sides of the critical factorization uv of w. If $|x| > |u|$, then observing that $|b| < |u|$ and letting $u = bv'$, where v' is a prefix of v, we get a contradiction since v' is consistent with both sides of the critical factorization uv of w. ■

We shall now prove the overlap-rotation lemma, which is the key to the improved approximation bounds. Given a semi-infinite string $\alpha = a_1 a_2 \cdots$, we denote the rotation $\alpha[k] = a_k a_{k+1} \cdots$.

Lemma 3.3 *Let α be a periodic semi-infinite string. There exists an integer k, such that for any (finite) string s that is inequivalent to α,*

$$ov(s, \alpha[k]) < period(s) + \frac{1}{2} period(\alpha).$$

In addition, if $period(s) \leq period(\alpha)$*, then*

$$ov(s, \alpha[k]) < \frac{2}{3}(period(s) + period(\alpha)).$$

Proof: We first show that there exists a suffix α' of α, such that the leftmost critical factorization $u\beta$ of $\alpha' = u\beta$ has the property that $|u| \leq \frac{1}{2}period(\alpha)$. We then prove that such a suffix α' satisfies the overlap requirement.

Let $x\alpha'$ be an arbitrary critical factorization of $\alpha = x\alpha'$ and let $w = factor(\alpha')$. Then it follows that w is unbordered, similarly to the first part of Lemma 3.2. Let uv be a critical factorization of $w = uv$. If $|u| \leq \frac{1}{2}period(\alpha)$, then $\alpha' = (uv)^\infty$ is the desired rotation, and otherwise, $|v| \leq \frac{1}{2}period(\alpha)$ and by Lemma 3.2, the rotation $(vu)^\infty$ satisfies the requirements.

If $period(s) = period(\alpha)$, then for any integer $k \geq 1$, the overlap $ov(s, \alpha[k]) < period(s)$. Otherwise, recalling that $\alpha' = u\beta$ is a critical factorization and $|u| \leq \frac{1}{2}period(\alpha)$, we claim that

$$ov(s, \alpha') < period(s) + |u| \leq period(s) + \frac{1}{2}period(\alpha).$$

To see this, assume on the contrary that $ov(s, \alpha') \geq period(s) + |u|$. But then, there is a string x of length $period(s)$ that is consistent with both sides of the critical factorization $u\beta$. If $period(s) < period(\alpha)$, then this immediately contradicts the fact that $u\beta$ is a critical factorization. If $period(s) > period(\alpha)$, then x has also a factor of length $period(\alpha)$, and therefore, $x = y^h z$, for some y such that $|y| = period(\alpha)$ and z proper prefix of y. If $|z| \geq 1$, then we obtain a contradiction since z is consistent with both sides of the critical factorization $u\beta$, and otherwise, if $|z| = 0$, then we contradict the fact the s and α were inequivalent.

If $period(s) \leq period(\alpha)$, then since $factor(\alpha')$ is unbordered, we have that $ov(s, \alpha') < period(\alpha)$. Putting the two inequalities together, we have

$$ov(s, \alpha') < \min\{period(s) + \frac{1}{2}period(\alpha), period(\alpha)\} \leq \frac{2}{3}(period(s) + period(\alpha)).$$

The proof above is constructive and requires two computations of critical factorizations, which can be done in time that is linear in $period(\alpha)$ as shown by Crochemore and Perrin [6, 7]. From now on, let $\overrightarrow{\alpha}$ denote a rotation of α satisfying Lemma 3.3, for any periodic semi-infinite string α. The bound in the last lemma is roughly tight because for any rotation of the semi-infinite string $(0^n 10^{n+1}1)^\infty$, there exists a string with period at most $n + 2$ which overlaps with $(0^n 10^{n+1}1)^\infty$ by at least $2n + 2$.

4 The generic approximation algorithm

Our algorithms are only slightly different from the ones in [1, 2, 3, 4, 8, 13, 21]. We first outline the general approach and then fill in the details of our new

1. Construct the distance graph G_S for set S.
2. Find a minimum weight cycle cover C on the graph G_S.
3. For each cycle $c = s_{i_1}, \ldots, s_{i_r}, s_{i_1} \in C$, choose a string t_c, such that for some j
   (i) t_c contains $\langle s_{i_{j+1}}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_j} \rangle$, and
   (ii) t_c is contained in $\langle s_{i_j}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_{j-1}}, s_{i_j} \rangle$.
4. Let T be the set of all strings chosen above and construct the distance graph G_T for T.
5. Find a minimum weight cycle cover CC on G_T.
6. Break each cycle of CC arbitrarily to obtain a superstring of the elements in the cycle.
7. Concatenate the strings found at Step 6 arbitrarily to produce a superstring $\tilde{s}$ of S.

Figure 2: The generic shortest superstring approximation algorithm.

constructions and analysis.

The main steps of the generic shortest superstring algorithm are shown in Figure 2. (A close variant of the generic algorithm has appeared in [1, 2].) A key difference between the above algorithm and all the previous algorithms is Step 3. The previous algorithms all choose one of the strings contained in the cycle c, whereas here we look for a superstring of the strings in c that is not *too long*. The string chosen does not even have to be one of the strings obtained by breaking c.

As a warm-up, let's show that this generic algorithm has approximation ratio 3. The following lemma is straightforward and is given in [4, 21]. Again, note that

$$\langle s_{i_j}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_{j-1}}, s_{i_j} \rangle = factor(\langle s_{i_j}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_{j-1}} \rangle) s_{i_j}.$$

Lemma 4.1 $\mathrm{opt}(T) \leq \mathrm{opt}(S) + \mathrm{CYC}(G_S) \leq 2\mathrm{opt}(S)$.

Hence, we have $\mathrm{CYC}(G_T) \leq \mathrm{opt}(T) \leq 2\mathrm{opt}(S)$. We now need a lemma which gives an upper bound on the possible overlap between two inequivalent strings. Different versions of the lemma in terms of discrete periodic functions or strings from distinct cycles in a minimum weight cycle cover can be found in [4, 10].

Lemma 4.2 *For any inequivalent strings s and t, $ov(s,t) \leq period(s) + period(t)$.*

The strings $\langle s_{i_{j+1}}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_j} \rangle$ and $\langle s_{i_j}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_{j-1}}, s_{i_j} \rangle$ are equivalent by Corollary 2.6, and thus, it follows from Lemma 2.1 and Corollary 2.5 that t_c is equivalent to $\langle s_{i_1}, \ldots, s_{i_r} \rangle$. Because C has the minimum weight, Lemma 2.7 further implies that the strings in set T are mutually inequivalent. Hence Lemma 4.2 applies to the strings in T. Let OV denote the total overlap represented by the edges broken in Step 6. Then OV is at most the sum of the periods of the strings in T. By Corollary 2.4,

$$OV \leq \sum_{c \in C} w(c) = \mathrm{CYC}(G_S).$$

Putting everything together, we can bound the length of the superstring $\tilde{s}$ as

$$|\tilde{s}| = \mathrm{CYC}(G_T) + OV \leq \mathrm{CYC}(G_T) + \mathrm{CYC}(G_S) \leq 2\mathrm{opt}(S) + \mathrm{opt}(S) \leq 3\mathrm{opt}(S).$$

5 The $2\frac{2}{3}$-approximation algorithm

Many researchers have tried to improve the performance of the generic algorithm or its variants by polishing Steps 5 - 7. Teng and Yao [21], Czumaj et al. [8], and Armen and Stein [1, 2] treat the small cycles (*i.e.* cycles with two or three vertices) in CC with special care. Teng and Yao [21] and Czumaj et al. [8] do so by finding a short path across the small cycles and Armen and Stein [1, 2] by identifying strings that are not much longer than their factors (called *short periodic strings* in their paper) as the bottleneck, and trying to avoid them in Step 3. Kosaraju et al. [13] find a Hamiltonian path with large overlap instead of CC in Steps 5-7.

Our algorithm is more similar to Armen and Stein's in the sense that we also choose the strings in Step 3 very carefully before going into the next round of cycle cover computation. (But we do not pay special attention to the small cycles.) The new idea is to choose strings which are guaranteed not to overlap with each other by too much. This will imply a reduced OV.

We now show how to choose the string t_c in Step 3 of the generic algorithm so that it satisfies the conditions (i) and (ii) and it has the correct rotation as prescribed by Lemma 3.3.

Lemma 5.1 *For any cycle $c = s_{i_1}, \ldots, s_{i_r}, s_{i_1} \in C$, there exists a string t such that for some j,*

1. *t contains the string $\langle s_{i_{j+1}}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_j} \rangle$.*
2. *t is contained in the string $\langle s_{i_j}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_{j-1}}, s_{i_j} \rangle$.*
3. *$t_\infty = \overrightarrow{\langle s_{i_1}, \ldots, s_{i_r} \rangle}_\infty$.*

Proof: Order the strings $\langle s_{i_1}, \ldots, s_{i_r} \rangle, \ldots, \langle s_{i_r}, s_{i_1}, \ldots, s_{i_{r-1}} \rangle$ according to their first appearances in $\overrightarrow{\langle s_{i_1}, \ldots, s_{i_r} \rangle}_\infty$. The ordering is unique. Suppose that $\langle s_{i_{j+1}}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_j} \rangle$ comes *first* in this ordering and let t be the prefix of

1.	Construct the distance graph G_S for set S.
2.	Find a minimum weight cycle cover C on the graph G_S.
3.	For each cycle $c \in C$, choose a string t_c as in Lemma 5.1.
4.	Let T be the set of all strings chosen above and construct the distance graph G_T.
5.	Find a minimum weight cycle cover CC on G_T.
6.	For each cycle of CC, break the cycle by deleting an edge that goes from a string to a string of *equal or larger period*, to obtain a superstring of the elements in the cycle.
7.	Concatenate the strings arbitrarily to produce a superstring $\tilde{s}$ of S.

Figure 3: The $2\frac{2}{3} \approx 2.67$-approximation algorithm.

$\overrightarrow{\langle s_{i_1}, \ldots, s_{i_r} \rangle_\infty}$ that ends at $\langle s_{i_{j+1}}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_j} \rangle$ (inclusive). Then t is contained in $\langle s_{i_j}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_{j-1}}, s_{i_j} \rangle$. Otherwise, $\langle s_{i_j}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_{j-1}} \rangle$ must appear before $\langle s_{i_{j+1}}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_j} \rangle$ in $\overrightarrow{\langle s_{i_1}, \ldots, s_{i_r} \rangle_\infty}$, which is a contradiction to the choice of $\langle s_{i_{j+1}}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_j} \rangle$. ∎

The string t chosen above will be denoted as t_c. We have shown how each t_c can be found in polynomial time (in fact, in linear time). We now polish the generic algorithm in Figure 3.

Note that we do not treat the small cycles of CC specially like the other algorithms do. Instead, we cut the cycles with a bit of care. Clearly, in every cycle there must be an edge that goes from a string to a string of equal or larger period.

Theorem 5.2 $|\tilde{s}| \leq 2\frac{2}{3}\mathrm{opt}(S) \approx 2.67\mathrm{opt}(S)$.

Proof: Again, let OV denote the total overlap lost by cutting the edges in Step 6. Since the strings in T are mutually inequivalent, by Lemma 3.3 and Corollary 2.4,

$$OV \leq \frac{2}{3} \sum_{c \in C} period(t_c) = \frac{2}{3} \sum_{c \in C} w(c) = \frac{2}{3}\mathrm{CYC}(G_S) \leq \frac{2}{3}\mathrm{opt}(S).$$

Hence, $|\tilde{s}| = \mathrm{CYC}(G_T) + OV \leq 2\frac{2}{3}\mathrm{opt}(S)$. ∎

6 The $2\frac{25}{42}$-approximation algorithm

The approach followed by the $2\frac{25}{42}$-approximation algorithm described in this section is very similar to that in [4, 13]. The main steps of the algorithm resemble the generic algorithm and are outlined in Figure 4. The cycle representatives t_c are chosen as in the previous section.

1.	Construct the distance graph G_S for set S.
2.	Find a minimum weight cycle cover C on the graph G_S.
3.	For each cycle $c = s_{i_1}, \ldots, s_{i_r}, s_{i_1} \in C$, choose a string t_c such that for some j (i) t_c contains $\langle s_{i_{j+1}}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_j} \rangle$, and (ii) t_c is contained in $\langle s_{i_j}, \ldots, s_{i_r}, s_{i_1}, \ldots, s_{i_{j-1}}, s_{i_j} \rangle$.
4.	Let T be the set of all strings chosen above. Construct a superstring of T using a good overlap approximation algorithm.

Figure 4: The $2\frac{25}{42} \approx 2.596$-approximation algorithm.

The following lemma is a close variant of a lemma proved in [4] and used in [13].

Lemma 6.1 *Let* apx(T) *be the length of the superstring of* T *produced by a* δ *overlap approximation algorithm. Then,*

$$\text{apx}(T) \leq \text{opt}(T) + (1-\delta)\text{maxov}(T).$$

Proof: Recall that $\text{opt}(T) = \sum_{t_i \in T} |t_i| - \text{maxov}(T)$. Since the overlap achieved by the δ overlap approximation algorithm is at least δ maxov(T), we get that

$$\text{apx}(T) \leq \sum_{t_i \in T} |t_i| - \delta \ \text{maxov}(T) = \text{opt}(T) + (1-\delta)\text{maxov}(T). \quad \blacksquare$$

We now need an upper bound on the possible overlap maxov(T). The standard bound used in all previous papers was $\text{maxov}(T) \leq 2\text{CYC}(G_S)$, which follows from Lemma 4.2. We show next that with our special choice of the cycle representatives t_c in Step 3, we can improve on this bound.

Lemma 6.2 $\text{maxov}(T) \leq \frac{3}{2}\text{CYC}(G_S)$.

Proof: Consider the shortest superstring for T, and assume that it contains $t_1, t_2, \ldots$, in this order. Recall that the strings in T are mutually inequivalent. Therefore, by Lemma 3.3,

$$ov(t_i, t_{i+1}) \leq period(t_i) + \frac{1}{2} period(t_{i+1}).$$

Summing over all strings $t_i \in T$, we get by Corollary 2.4 that,

$$\text{maxov}(T) = \sum_{i=1}^{|T|-1} ov(t_i, t_{i+1}) \leq \frac{3}{2} \sum_{t_i \in T} period(t_i) = \frac{3}{2}\text{CYC}(G_S). \quad \blacksquare$$

Putting everything together, and using the $\frac{38}{63}$ overlap approximation algorithm of Kosaraju et al. [13] in Step 4, we establish the following theorem.

Theorem 6.3 *The algorithm in Figure 4 is a* $2\frac{25}{42} \approx 2.596$*-approximation for the shortest superstring problem.*

Proof: By Lemmas 4.1, 6.1 and 6.2, the superstring produced by the algorithm has length

$$\text{apx}(T) \leq \text{opt}(T) + (1 - \frac{38}{63})\text{maxov}(T) \leq 2\text{opt}(S) + \frac{25}{63}\frac{3}{2}\text{CYC}(G_S) \leq 2\frac{25}{42}\,\text{opt}(S).$$

7 Concluding Remarks

We are still a long way from reaching the conjectured ratio 2 for approximating shortest superstrings.

Acknowledgement. We thank Maxime Crochemore and Bill Smyth for many helpful discussions on string combinatorics and for providing useful references.

Maxime Crochemore suggested an alternative simple proof of Lemma 3.3 without using the Critical Factorization Theorem. His proof uses properties of Lyndon words directly, in the spirit of the recent proofs of the Critical Factorization Theorem in [6, 7], and exploiting the fact that the string in question α is infinite (or long relatively to its period). In particular, fixing an arbitrary total order on the different symbols of α, if $(xy)^\infty$ and $(yx)^\infty$ are the lexicographically minimal and maximal rotations of α, then it easily follows that x–$(yx)^\infty$ and y–$(xy)^\infty$ are critical factorizations and that both rotations begin with an unbordered factor.

References

[1] C. Armen and C. Stein. Improved Length Bounds for the Shortest Superstring problem. In *Proc. 4th Workshop on Algorithms and Data Structures*, number 955 in Lecture Notes in Computer Science, pages 494–505. Springer-Verlag, Berlin, Germany, 1995.

[2] C. Armen and C. Stein. Short superstrings and the structure of overlapping strings. Manuscript, 1995.

[3] C. Armen and C. Stein. A $2\frac{2}{3}$-Approximation Algorithm for the Shortest Superstring Problem. In *Proc. 7th Symp. on Combinatorial Pattern Matching*, Lecture Notes in Computer Science, page to appear. Springer-Verlag, Berlin, Germany, 1996.

[4] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yanakakis. Linear Approximation of Shortest Superstrings. *J. Assoc. Comput. Mach.*, 41(4):630–647, 1994.

[5] Y. Césari and M. Vincent. Une caractérisation des mots périodiques. *C.R. Acad. Sci. Paris*, 286(A):1175–1177, 1978.

[6] M. Crochemore and D. Perrin. Two-way string-matching. *J. Assoc. Comput. Mach.*, 38(3):651–675, 1991.

[7] M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.

[8] A. Czumaj, L. Gąsieniec, M. Piotrow, and W. Rytter. Parallel and Sequential Approximations of Shortest Superstrings. In *Proc. 4th Scandinavian Workshop on Algorithm Theory*, number 824 in Lecture Notes in Computer Science, pages 95–106. Springer-Verlag, Berlin, Germany, 1995.

[9] R. Fleischmann et al. Whole-genome random sequencing and assembly of *Haemophilus influenzae* Rd. *Science* 269, 496-512, July 1995.

[10] N.J. Fine and H.S. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.*, 16:109–114, 1965.

[11] J. Gallant, D. Maier, and J. Storer. On Finding Minimal Length Superstrings. *J. Comput. System Sci.*, 20:50–58, 1980.

[12] T. Jiang and M. Li. DNA sequencing and string learning. *Math. Systems Theory*, 1993. To appear.

[13] S.R. Kosaraju, J. Park, and C. Stein. Long Tours and Short Superstrings. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, 1994.

[14] A. Lesk, editor. *Computational Molecular Biology, Sources and Methods for Sequence Analysis.* Oxford University Press, 1988.

[15] M. Li. Toward a DNA sequencing theory. In *Proc. 31th IEEE Symp. on Foundations of Computer Science*, pages 125–134, 1990.

[16] M. Lothaire. *Combinatorics on Words.* Addison-Wesley, Reading, MA, U.S.A., 1983.

[17] C Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[18] H. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen. Algorithms for some string matching problems arising in molecular genetics. In *Information Processing 83 (Proc. IFIP Congress)*, pages 53–64, 1973.

[19] J. Storer. *Data Compression: Methods and Theory.* Addison-Wesley, 1988.

[20] J. Tarhio and E. Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theoret. Comput. Sci.*, 57:131–145, 1988.

[21] S.H. Teng and F. Yao. Approximating Shortest Superstrings. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 158–165, 1993.

[22] J. Turner. Approximation Algorithms for the Shortest Common Superstring Problem. *Information and Computation*, 83:1–20, 1989.

[23] M.S. Waterman. *Introduction to Computational Biology: Maps, Sequences, and Genoms (Interdisciplinary Statistics).* Chapman and Hall, 1995.

Recent Publications in the BRICS Report Series

RS-96-21 **Dany Breslauer, Tao Jiang, and Zhigen Jiang.** ***Rotation of Periodic Strings and Short Superstring*****. June 1996. 14 pp.**

RS-96-20 **Olivier Danvy and Julia L. Lawall.** ***Back to Direct Style II: First-Class Continuations*****. June 1996. 36 pp. A preliminary version of this paper appeared in the proceedings of the 1992 ACM Conference on Lisp and Functional Programming, William Clinger, editor, LISP Pointers, Vol. V, No. 1, pages 299–310, San Francisco, California, June 1992. ACM Press.**

RS-96-19 **John Hatcliff and Olivier Danvy.** ***Thunks and the λ-Calculus*****. June 1996. 22 pp. To appear in** ***Journal of Functional Programming*****.**

RS-96-18 **Thomas Troels Hildebrandt and Vladimiro Sassone.** ***Comparing Transition Systems with Independence and Asynchronous Transition Systems*****. June 1996. 14 pp. To appear in*****Concurrency Theory: 7th International Conference*****, CONCUR '96 Proceedings, LNCS, 1996.**

RS-96-17 **Olivier Danvy, Karoline Malmkjær, and Jens Palsberg.** ***Eta-Expansion Does The Trick (Revised Version)*****. May 1996. 30 pp. To appear in** ***ACM Transactions on Programming Languages and Systems (TOPLAS)*****.**

RS-96-16 **Lisbeth Fajstrup and Martin Raußen.** ***Detecting Deadlocks in Concurrent Systems*****. May 1996. 10 pp.**

RS-96-15 **Olivier Danvy.** ***Pragmatic Aspects of Type-Directed Partial Evaluation*****. May 1996. 27 pp.**

RS-96-14 **Olivier Danvy and Karoline Malmkjær.** ***On the Idempotence of the CPS Transformation*****. May 1996. 15 pp.**

RS-96-13 **Olivier Danvy and René Vestergaard.** ***Semantics-Based Compiling: A Case Study in Type-Directed Partial Evaluation*****. May 1996. 28 pp.**