Electronic Colloquium on Computational Complexity, Report No. 157 (2026)

A Tight Cycle-Cover Inequality for Shortest Common Superstring

Nikolai Chukhin * Alexander S. Kulikov † Ivan Mihajlin ‡ Alexander Smal §

Abstract

In the Shortest Common Superstring problem (SCS), one is given a finite set of strings and is asked to find a shortest string containing every input string as a substring. Its best known approximation ratio is 2.466, whereas the currently strongest upper bound on the approximation guarantee of the maximum-overlap greedy algorithm is 3.396 (Englert, Matsakis, and Veselý, 2023), though it is conjectured to be 2. We improve both approximation guarantees: SCS admits a $\frac{7}{3}$-approximation and the approximation guarantee of the greedy algorithm is at most 3.

The main technical ingredient of our proof is a certain inequality for optimum cycle covers of an overlap graph associated with the input strings. Every previous improvement of greedy's worst-case guarantee and the two recent record guarantees for general SCS are driven by it. We improve this inequality by pushing it to its limit: for a particular coefficient of this inequality, we show a new upper bound and prove that it cannot be improved further.

1 Approximating Shortest Common Superstring

In the Shortest Common Superstring problem (SCS), one is given a finite set of strings and is asked to find a shortest string containing every input string as a substring. The problem is a classical model of assembling highly overlapping data and appears in data compression and genome assembly [GP14]. It is NP-hard already for strings of length three [GMS80], and, for this reason, polynomial-time approximation algorithms are actively studied. The first constant-factor algorithm, with approximation ratio 3, was given in [BJL+91]. Subsequent work improved this ratio through increasingly delicate combinations of cycle covers and maximum asymmetric traveling salesperson algorithms. The currently best known approximation ratio is 2.466 [EMV23]. Interestingly, there is a simple greedy algorithm whose conjectured [TU88, Sto88, Tur89, BJL+91] approximation ratio is equal to 2: while more than one string remains, choose an ordered pair with maximum suffix–prefix overlap, merge the pair, and iterate. This algorithm admits a linear-time implementation over a constant-size alphabet [Ukk90] and is known to behave well in practice [RBT04, Ma09, SVB23]. Whereas it is not difficult to show that its approximation ratio is at least 2, the best known upper bound is 3.396 [EMV23].

Our Contribution

We improve both approximation guarantees: SCS admits a $\frac{7}{3}$-approximation and the approximation guarantee of the greedy algorithm is at most 3. In Figure 1, we show a timeline of all known upper bounds on the approximation ratio of SCS and the greedy algorithm.

*JetBrains Research. Email: buyolitsez1951@gmail.com
†JetBrains Research. Email: alexander.s.kulikov@gmail.com
‡JetBrains Research. Email: ivmihajlin@gmail.com
§JetBrains Research. Email: avsmal@gmail.com

ISSN 1433-8092

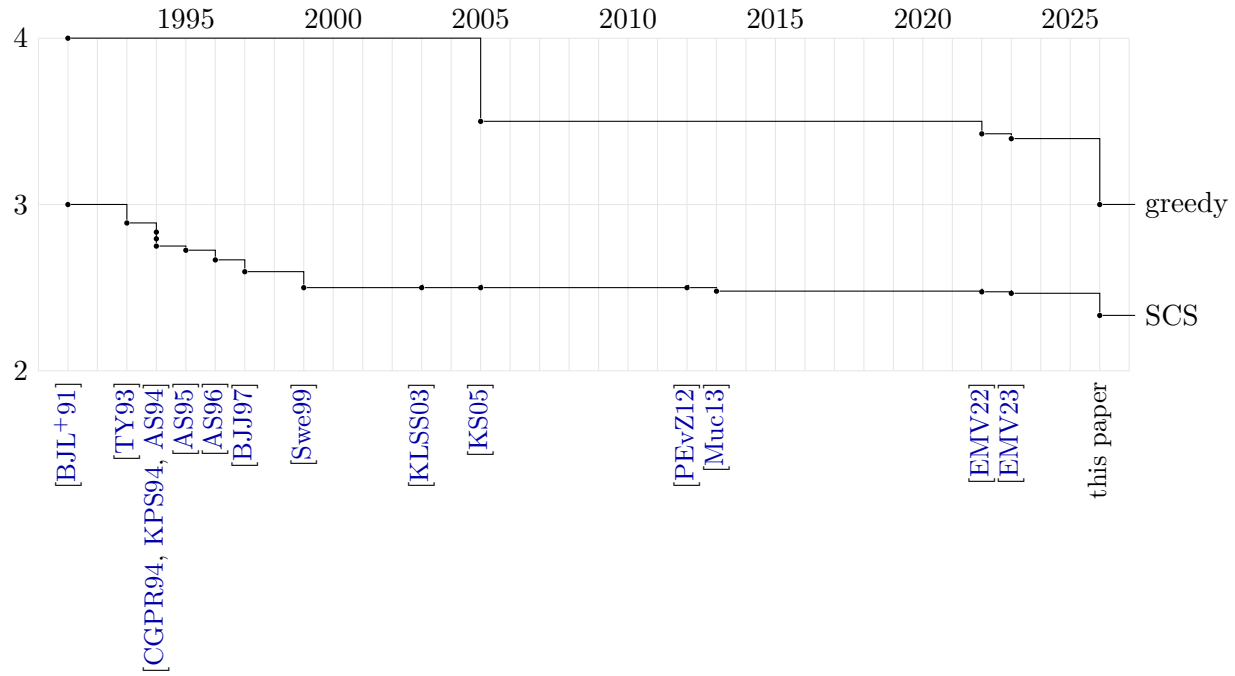


Figure 1: Timeline of published approximation ratios for SCS (bottom curve) and the greedy algorithm for SCS (top curve).

2 Known Technical Results

We start by listing important known results on SCS; we introduce related concepts along the way. Gray boxes are used to visualize these concepts and facts; they are intended to help the reader and may be safely skipped.

In the rest of the paper, $\mathcal{S} = \{s_1, \ldots, s_n\}$ is an input to SCS; we assume that no string in $\mathcal{S}$ is a substring of another string from $\mathcal{S}$ (one can get rid of such cases on the preprocessing stage). By OPT we denote an optimal common superstring of $\mathcal{S}$ or its length.

2.1 Prefixes and Overlaps

For nonempty strings s and t, their *overlap* $\mathrm{ov}(s,t)$ is the longest (possibly empty) string q such that $s = p \circ q$ and $t = q \circ r$, for some *nonempty* strings p and r that are in turn denoted by $\mathrm{pref}(s,t)$ and $\mathrm{suff}(s,t)$. For example, $\mathrm{ov}(\mathtt{ababa}, \mathtt{babc}) = \mathtt{ba}$ and $\mathrm{ov}(\mathtt{ababa}, \mathtt{ababa}) = \mathtt{aba}$. By $\langle s,t \rangle$ we denote the *overlap merge* of s and t:

$$\langle s,t \rangle = s \circ \mathrm{suff}(s,t) = \mathrm{pref}(s,t) \circ t = \mathrm{pref}(s,t) \circ \mathrm{ov}(s,t) \circ \mathrm{suff}(s,t).$$

Although the overlap and the prefix are defined as strings, we use these two words to denote the lengths of the corresponding strings. The notion of the overlap merge extends to more than two strings in a natural way: for $t_1, \ldots, t_k$,

$$\langle t_1, \ldots, t_k \rangle = \mathrm{pref}(t_1, t_2) \circ \mathrm{pref}(t_2, t_3) \circ \cdots \circ \mathrm{pref}(t_{k-1}, t_k) \circ t_k.$$

Clearly,

$$|\langle t_1, \ldots, t_k \rangle| = \sum_{i \in [k-1]} |\mathrm{pref}(t_i, t_{i+1})| + |t_k| = \sum_{i \in [k]} |t_i| - \sum_{i \in [k-1]} |\mathrm{ov}(t_i, t_{i+1})|.$$

Hence, the smaller the sum of adjacent prefixes, the larger the sum of adjacent overlaps.

2.2 Permutations and Related Graphs

The goal in the SCS problem is to find a permutation of $\mathcal{S}$ whose overlap merge length is minimum. This is equivalent to maximizing the total overlap or minimizing the total prefix (up to the length of the last string in the permutation). The greedy algorithm tries to construct a permutation with large total overlap by taking, at every step, the largest available overlap.

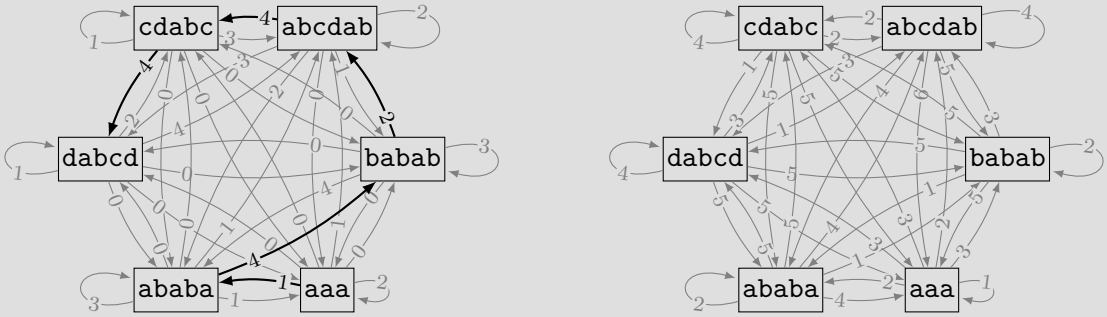
For a set of strings $\{\texttt{abcdab}, \texttt{cdabc}, \texttt{dabcd}, \texttt{ababa}, \texttt{aaa}, \texttt{babab}\}$, a naive superstring results from concatenating them and has length 6+5+5+5+3+5 = 29: `abcdabcdabcdabcdababaaaababab`. An optimal superstring has length 14: `aaababababcdabcd`. It corresponds to a permutation $(\texttt{aaa}, \texttt{ababa}, \texttt{babab}, \texttt{abcdab}, \texttt{cdabc}, \texttt{dabcd})$. Its total overlap is $1 + 4 + 2 + 4 + 4 = 15$, exactly the number of symbols saved relative to the naive concatenation of length 29.
The largest overlap is 4; therefore, the greedy algorithm may start by merging $\langle \texttt{dabcd}, \texttt{abcdab}, \texttt{cdabc} \rangle$, after which it may merge $\langle \texttt{babab}, \texttt{ababa} \rangle$. Afterwards, there are no overlaps larger than 1, and the algorithm may finish with the following superstring of length 16:

$$\langle \texttt{babab}, \texttt{ababa}, \texttt{aaa}, \texttt{dabcd}, \texttt{abcdab}, \texttt{cdabc} \rangle = \texttt{bababaaadabcdabc}.$$

All pairwise overlaps between input strings can be conveniently represented in an *overlap graph*: it is a complete directed graph with vertices $\mathcal{S}$ and edges $\mathcal{S} \times \mathcal{S}$ (thus, self-loops are included) where the length of an edge (s, t) is $|\mathrm{ov}(s, t)|$. A *prefix graph* is defined similarly with the only difference that the lengths are $|\mathrm{pref}(s, t)|$.

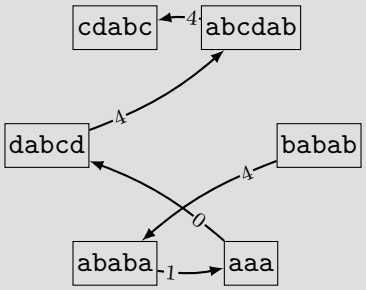
Solving SCS is the same as solving MAX-ATSP in the overlap graph, that is, finding a Hamiltonian path of maximum total length. At the same time, approximation algorithms for MAX-ATSP do not give approximation guarantees for SCS directly since the total overlap may be much larger than OPT.

The overlap and prefix graphs for our running example are shown below. In the overlap graph, a Hamiltonian path of maximum total overlap 15 is highlighted.



The greedy superstring algorithm can be conveniently represented in terms of the overlap graph: it constructs a Hamiltonian path by scanning the list of edges in the order of nonincreasing length and takes the current edge into a solution if it does not create a node of in-degree or out-degree more than one and does not close a cycle. This way, at any point of the execution of the greedy algorithm, its solution is a collection of disjoint paths.

The greedy algorithm starts by considering edges of length 4. It may take edges `dabcd` → `abcdab` and `abcdab` → `cdabc`. Then, it skips the edge `cdabc` → `dabcd` as it would create a cycle. The greedy algorithm then takes an edge `babab` → `ababa`. Finally, by skipping a bunch of edges, it takes edges `ababa` → `aaa` and `aaa` → `dabcd`.



2.3 Cycle Covers

If one allows the greedy algorithm to produce cycles (without violating the in-degree and out-degree constraints), then it constructs an optimal *cycle cover* $\mathcal{C}$, that is, a collection of cycles that cover every node of the graph and has maximum total overlap. This is proved in [BJL+91, Theorem 10]. The constructed cycle cover may contain self-loops. It is not difficult to see that $\mathcal{C}$ is also a cycle cover of minimum total length in the prefix graph. Indeed, consider a cycle $C = (t_0, t_1, \ldots, t_{r-1}, t_0)$ of $\mathcal{C}$. Its total overlap is equal to $\sum_{i\in[r]} |\mathrm{ov}(t_{i-1}, t_i)|$ and its total prefix is $\sum_{i\in[r]} |\mathrm{pref}(t_{i-1}, t_i)|$ (here, indices are taken modulo r). Note, however, that $|\mathrm{pref}(t_{i-1}, t_i)| + |\mathrm{ov}(t_{i-1}, t_i)| = |t_{i-1}|$. Consequently, the total overlap of C and its total prefix sum to the total length of the strings on C. Thus, maximizing total overlap minimizes total prefix.

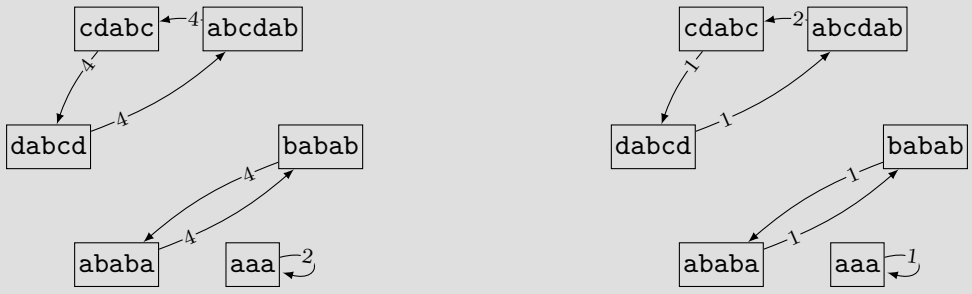
For $C \in \mathcal{C}$, let $w(C)$ be the total prefix of the cycle C and $W = \sum_{C\in\mathcal{C}} w(C)$ be the total prefix of $\mathcal{C}$. Then,

$$W \le \mathrm{OPT}\,. \tag{1}$$

Indeed, let $(t_1, \ldots, t_n)$ be a permutation of the input strings whose overlap merge has length OPT. Then, the total prefix of the cycle $(t_1, \ldots, t_n, t_1)$ is at most OPT, but at least W (since this single cycle is also a cycle cover):

$$\mathrm{OPT} = \sum_{i\in[n-1]} |\mathrm{pref}(t_i, t_{i+1})| + |t_n| \ge \sum_{i\in[n-1]} |\mathrm{pref}(t_i, t_{i+1})| + |\mathrm{pref}(t_n, t_1)| \ge W.$$

The figure below shows the same optimal cycle cover in the overlap and prefix graphs. Its total overlap and total prefix are 22 and $W = 7$, respectively; their sum is 29, the total length of the input strings.



2.4 Opening the Cycles

To turn a cycle cover into a Hamiltonian path, one may proceed as follows: for each cycle, remove its edge of minimum overlap; then, merge the resulting paths into a Hamiltonian path.

Formally, consider a cycle $C = (t_1, \ldots, t_r, t_1)$ of $\mathcal{C}$ and assume that the edge (t_r, t_1) was the last edge of this cycle added by the algorithm. We call it a *cycle-closing edge*; it has minimum overlap among the edges of the cycle. Denote its length by $o(C)$ and let $O = \sum_{C \in \mathcal{C}} o(C)$. Now, let $\sigma(C) = \langle t_1, \ldots, t_r \rangle$ be a *representative string* for C. Then,

$$\sigma(C) = \mathrm{pref}(t_1, t_2) \circ \cdots \circ \mathrm{pref}(t_{r-1}, t_r) \circ t_r = \mathrm{pref}(t_1, t_2) \circ \cdots \circ \mathrm{pref}(t_{r-1}, t_r) \circ \mathrm{pref}(t_r, t_1) \circ \mathrm{ov}(t_r, t_1).$$

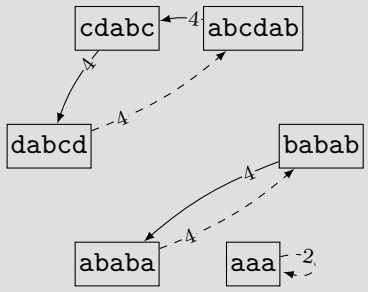
Hence,

$$|\sigma(C)| = w(C) + o(C).$$

An algorithm that constructs an optimal cycle cover and returns a concatenation is called MGREEDY; it was introduced in [BJL+91]. The resulting superstring has length

$$\sum_{C \in \mathcal{C}} (w(C) + o(C)) = W + O. \tag{2}$$

The cycle-closing edges of the optimal cycle cover are shown as dashed edges below. Their total overlap is $O = 4 + 4 + 2 = 10$. To the right, we show the corresponding representative strings. By concatenating the representative strings, one gets a superstring `abcdabcdbababaaaa` of length $W + O = 7 + 10 = 17$.



$\sigma(C_1) =$ `abcdabcd`
$\sigma(C_2) =$ `bababa`
$\sigma(C_3) =$ `aaa`

2.5 Upper Bounding the Overlap of Cycle-Closing Edges

Since we already know that $W \leq \mathrm{OPT}$ (see (1)), to turn an upper bound $W + O$ (see (2)) into an approximation guarantee, we need to upper bound O in terms of OPT. This is exactly what many previous papers do. Namely, they prove an upper bound of the form

$$O \leq \mathrm{OPT} + \gamma W, \tag{3}$$

for a constant γ. Combined with (1) and (2), this inequality immediately implies that MGREEDY is a $(2 + \gamma)$-approximation. As shown by [BJL+91, Section 5], this implies that the approximation guarantee of the greedy algorithm is also at most $(2 + \gamma)$.

The MGREEDY algorithm simply concatenates the representative strings. If, instead, one applies an α-approximation algorithm of MAX-ATSP in the overlap graph of the representative strings and merges them in the found order, the resulting algorithm has approximation guarantee $(2 + (1 - \alpha)\gamma)$. This was shown in [BJJ97, Muc07, Muc13], a proof can also be found in [EMV22, Theorem 3.1]. The history of improvements of upper bounds on γ is the following: [BJL+91] proved

the inequality with $\gamma = 2$, [KS05] improved it to 3/2, [EMV22] pushed it to 1.425, and [EMV23] improved it further to 1.396.

We prove the inequality (3) for $\gamma = 1$ (see Theorem 3.1). This implies that the greedy algorithm is 3-approximate. Plugging in the best known approximation guarantee $\alpha = 2/3$ [KLSS03, PEvZ12] for MAX-ATSP gives a polynomial-time 7/3-approximation for SCS. Our proof of a stronger upper bound on γ does not come at a cost of a more involved analysis: in fact, our proof is much simpler. We achieve this by considering global properties of the overlap graph instead of considering each cycle of the optimal cycle cover separately. Then, in Theorem 3.3, we prove that $\gamma = 1$ is the optimal endpoint of this approach: for every $\gamma < 1$, there exists an input to SCS where the inequality (3) fails.

2.6 Properties of Cycle Strings

A cycle with small total prefix may be an indicator of the high repetitive structure of the strings of this cycle. For example, a cycle through strings

$$\texttt{abcdabcdabcd}, \texttt{cdabcdabcdab}, \texttt{dabcdabcdabc}$$

has total prefix $2 + 1 + 1 = 4$, whereas its minimum edge overlap is 10. In particular, O may be much larger than W.

Two strings are *equivalent* if one is a cyclic rotation of the other. A string z is *primitive* if there is no string v and an integer $k \geq 2$ such that $z = v^k$. For a cycle $C = (t_1, \ldots, t_r, t_1)$ of $\mathcal{C}$, define its *cycle string* by

$$\tau(C) = \operatorname{pref}(t_1, t_2) \circ \cdots \circ \operatorname{pref}(t_r, t_1),$$

hence $|\tau(C)| = w(C)$.

Lemma 2.1 ([Muc13, Lemma 2.5])**.** *Every cycle string $\tau(C)$ is primitive, and the cycle strings of distinct cycles of $\mathcal{C}$ are pairwise non-equivalent.*

In the running example, the three cycles C_1, C_2, C_3 have the following cycle strings:

$$\tau(C_1) = \texttt{abcd}, \qquad \tau(C_2) = \texttt{ba}, \qquad \tau(C_3) = \texttt{a}.$$

Then, $\sigma(C_1) = \tau(C_1)^2, \sigma(C_2) = \tau(C_2)^3$, and $\sigma(C_3) = \tau(C_3)^3$ (in general, $\sigma(C)$ is not always a power of $\tau(C)$).

Fix an arbitrary total order on the alphabet. A *Lyndon word* [Shi53, Lyn54] is a string that is strictly smaller than each of its proper suffixes; equivalently, it is primitive and lexicographically smaller than all its nontrivial cyclic rotations. Every equivalence class of primitive strings has a unique Lyndon word. Moreover, a Lyndon word z has no period $p < |z|$: otherwise its suffix of length $|z| - p$ would equal a proper prefix of z and hence would be smaller than z, contrary to the definition.

For a string x and an integer $m \geq 0$, let $\operatorname{Sub}_m(x)$ be the set of length-m substrings of x. For a string z and an integer $m \geq 0$, let $\operatorname{CSub}_m(z)$ be the set of length-m substrings of z^∞. For every cycle $C \in \mathcal{C}$, let $\tau'(C)$ be the unique Lyndon word equivalent to $\tau(C)$.

Lemma 2.2 ([BJL+91, Claim 2])**.** *For every cycle $C \in \mathcal{C}$,*

$$\operatorname{CSub}_{o(C)+1}(\tau'(C)) = \operatorname{CSub}_{o(C)+1}(\tau(C)) \subseteq \operatorname{Sub}_{o(C)+1}(\mathrm{OPT}).$$

Consider the optimal superstring OPT = aaababababcdabcd. Using the alphabet order a < b < c < d, the Lyndon words corresponding to the three cycle strings are

$$\tau'(C_1) = \texttt{abcd}, \qquad \tau'(C_2) = \texttt{ab}, \qquad \tau'(C_3) = \texttt{a}.$$

The minimum overlaps of the three cycles are $o(C_1) = o(C_2) = 4$ and $o(C_3) = 2$. Then,

$$\begin{aligned}
\mathrm{CSub}_5(\tau'(C_1)) &= \{\texttt{abcda}, \texttt{bcdab}, \texttt{cdabc}, \texttt{dabcd}\},\\
\mathrm{CSub}_5(\tau'(C_2)) &= \{\texttt{ababa}, \texttt{babab}\},\\
\mathrm{CSub}_3(\tau'(C_3)) &= \{\texttt{aaa}\}.
\end{aligned}$$

Thus, every length-$(o(C)+1)$ substring of $\tau(C)^\infty$ occurs as a substring of OPT.

3 Optimal Bound on the Cycle-Cover Inequality

In this section, we give a proof of our main technical contribution: the smallest value of γ where the inequality (3) holds is equal to 1.

3.1 Upper Bound

Theorem 3.1.

$$O \le \mathrm{OPT} + W.$$

Lemma 2.2 shows that, for every cycle C, all length-$(o(C)+1)$ substrings of $\tau(C)^\infty$ occur in OPT. The lemma below bounds the number of cycles that can use the same superstring.

Lemma 3.2. *Let u be a string, $m \in \{1, \dots, |u|\}$, and let $z_1, \dots, z_a$ be distinct Lyndon words for some $a \ge 0$. Suppose that $|z_j| \le m$ and $\mathrm{CSub}_m(z_j) \subseteq \mathrm{Sub}_m(u)$ for every j. Then*

$$a \le |\mathrm{Sub}_m(u)| - |\mathrm{Sub}_{m-1}(u)| + 1.$$

Consider the difference between Sub_m and Sub_{m-1}. Every distinct length-$(m-1)$ substring of u, except possibly the one occurring only at the end of u, can be extended to the right by one symbol. Therefore

$$|\,\mathrm{Sub}_m(u)| \ge |\,\mathrm{Sub}_{m-1}(u)| - 1.$$

Lemma 3.2 shows that each Lyndon word whose complete set of length-m periodic substrings occurs in u forces one additional length-m substring. To give an example, let $u =$ aababbabaab and $m = 4$. Then

$$\begin{aligned}
\mathrm{Sub}_3(u) &= \{\texttt{aab}, \texttt{aba}, \texttt{abb}, \texttt{baa}, \texttt{bab}, \texttt{bba}\},\\
\mathrm{Sub}_4(u) &= \{\texttt{aaba}, \texttt{abaa}, \texttt{abab}, \texttt{abba}, \texttt{baab}, \texttt{baba}, \texttt{babb}, \texttt{bbab}\}.
\end{aligned}$$

By the extension argument above, $|\,\mathrm{Sub}_4(u)| \ge |\,\mathrm{Sub}_3(u)| - 1 = 5$.
However, consider the Lyndon words $z_1 =$ aab, $z_2 =$ ab, and $z_3 =$ abb. Their length-4 cyclic substrings are

$$\begin{aligned}
\mathrm{CSub}_4(z_1) &= \{\texttt{aaba}, \texttt{abaa}, \texttt{baab}\},\\
\mathrm{CSub}_4(z_2) &= \{\texttt{abab}, \texttt{baba}\},\\
\mathrm{CSub}_4(z_3) &= \{\texttt{abba}, \texttt{babb}, \texttt{bbab}\}.
\end{aligned}$$

They are contained in $\mathrm{Sub}_4(u)$, so Lemma 3.2 implies that these three Lyndon words account for three additional length-4 substrings, which shows that

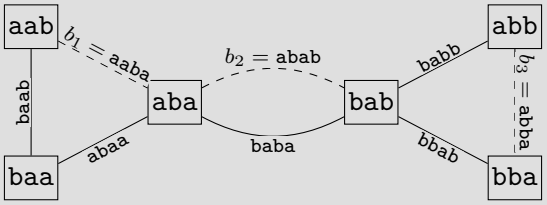
$$|\mathrm{Sub}_4(u)| \geq a + |\mathrm{Sub}_3(u)| - 1 = 8.$$

Proof. Let G_m be the following undirected multigraph: its vertices are the strings in $\mathrm{Sub}_{m-1}(u)$, and each string $\mathrm{Sub}_m(u)$ is an edge joining its length-$(m-1)$ prefix and its length-$(m-1)$ suffix (it is an undirected version of the order-$(m-1)$ Rauzy graph [Rau82] of u, which is in turn a subgraph of the de Bruijn graph [DB46]). Thus, G_m has $|\mathrm{Sub}_{m-1}(u)|$ vertices and $|\mathrm{Sub}_m(u)|$ edges. It is connected because the consecutive length-$(m-1)$ substrings of u form a walk that visits every vertex.

For every j, let b_j be the length-m prefix of z_j^∞. Since $m \geq |z_j|$ and z_j is a Lyndon word, b_j is the unique lexicographically smallest string in $\mathrm{CSub}_m(z_j)$. The strings $b_1, \ldots, b_a$ are pairwise distinct. Indeed, suppose $b_i = b_j$. If $|z_i| = |z_j|$, since $m \geq |z_i|$, the length-$|z_i|$ prefixes of b_i and b_j are z_i and z_j, respectively, so $z_i = z_j$. If, say, $|z_i| < |z_j|$, then z_j is a prefix of z_i^∞ and hence has period $|z_i| < |z_j|$, which is impossible for a Lyndon word. After relabeling, assume $b_1 < b_2 < \cdots < b_a$ and delete the edges $b_1, b_2, \ldots, b_a$ from G_m in that order. We show that after each deletion the graph remains connected.

Consider the deletion of the edge b_j, and, for $0 \leq i < |z_j|$, let c_i and u_i be the length-m and length-$(m-1)$ substrings of z_j^∞ beginning at offset i, respectively. Then $c_0 = b_j$, and each c_i is an edge of G_m connecting u_i and u_{i+1}. If $|z_j| > 1$, deleting the edge $c_0 = b_j$ keeps the graph connected, since the alternate walk $c_1, c_2, \ldots, c_{|z_j|-1}$ is still present: each c_i with $1 \leq i < |z_j|$ is larger than b_j, whereas every previously deleted edge is smaller than b_j. If $|z_j| = 1$, then b_j is a loop, so its deletion is harmless.

For the same string $u = \texttt{aababbabaab}$ and $m = 4$, the graph G_4 is shown below.



After all a deletions, the remaining graph is connected and has $|\mathrm{Sub}_m(u)| - a$ edges; consequently, $|\mathrm{Sub}_m(u)| - a \geq |\mathrm{Sub}_{m-1}(u)| - 1$. □

Proof of Theorem 3.1. Enumerate the cycles $C \in \mathcal{C}$ satisfying $o(C) + 1 \geq w(C)$ as $C_1, \ldots, C_h$. For every index j, put $z_j = \tau'(C_j)$ and $\ell_j = o(C_j) + 1$. The Lyndon words $z_1, \ldots, z_h$ are distinct by Lemma 2.1, and $|z_j| = |\tau(C_j)| = w(C_j) \leq \ell_j$ for every j. Moreover, Lemma 2.2 gives $\mathrm{CSub}_{\ell_j}(z_j) \subseteq \mathrm{Sub}_{\ell_j}(\mathrm{OPT})$ for every j. For $1 \leq m \leq \mathrm{OPT}$, let a_m be the number of indices j satisfying $|z_j| \leq m \leq \ell_j$. Lemma 3.2 gives

$$a_m \leq |\mathrm{Sub}_m(\mathrm{OPT})| - |\mathrm{Sub}_{m-1}(\mathrm{OPT})| + 1.$$

Each index j is counted by a_m for the values $m = |z_j|, \ldots, \ell_j$, hence exactly $\ell_j - |z_j| + 1 = o(C_j) - w(C_j) + 2$ times. Since $|\mathrm{Sub}_0(\mathrm{OPT})| = |\mathrm{Sub}_{\mathrm{OPT}}(\mathrm{OPT})| = 1$, summing the inequalities

over m gives

$$\begin{aligned}\sum_{\substack{C\in\mathcal{C}\\ o(C)+1\geq w(C)}} (o(C)-w(C)+2) &= \sum_{j=1}^{h}(\ell_j - |z_j|+1)\\ &= \sum_{m=1}^{\mathrm{OPT}} a_m\\ &\leq \sum_{m=1}^{\mathrm{OPT}} \left(|\mathrm{Sub}_m(\mathrm{OPT})| - |\mathrm{Sub}_{m-1}(\mathrm{OPT})|+1\right)\\ &= \mathrm{OPT}\,.\end{aligned}$$

For each cycle in the first sum, $o(C)-w(C)\leq o(C)-w(C)+2$, whereas every omitted cycle satisfies $o(C)-w(C)<0$. Therefore,

$$O - W \leq \mathrm{OPT}\,. \qquad \square$$

3.2 Lower Bound

In this section, we prove that the coefficient of W in Theorem 3.1 cannot be decreased further.

Theorem 3.3. *For every $\gamma<1$, there is a substring-free set $\mathcal{S}$ of binary strings such that*

$$O > \mathrm{OPT} + \gamma W.$$

Proof. For $p\geq 2$, put $s_p = \mathtt{0}^{p-1}\mathtt{1}\mathtt{0}^{p-1}\mathtt{1}\mathtt{0}^{p-2}$.

$$\begin{aligned} s_2 &= \mathtt{0101}\\ s_3 &= \mathtt{0010010}\\ s_4 &= \mathtt{0001000100}\\ s_5 &= \mathtt{0000100001000}\end{aligned}$$

Clearly, s_p starts and ends with $\mathtt{0}^{p-1}\mathtt{1}\mathtt{0}^{p-2}$, so $|\mathrm{ov}(s_p,s_p)|\geq 2p-2$. Also, for $p\neq q$, s_p is not a substring of s_q (s_p has exactly two $\mathtt{1}$'s, whose positions differ by p) and $|\mathrm{ov}(s_p,s_q)|\leq 2p-2$ (a longer overlap would cover both $\mathtt{1}$'s in s_p).

For $m\geq 2$, let $\mathcal{S}_m=\{s_2,s_3,\ldots,s_m\}$. By the discussion above, $\mathcal{S}_m$ is substring-free and its cycle cover consisting of all self-loops has minimum total prefix. Then,

$$\begin{aligned} W &= \sum_{p=2}^{m}(3p-2-(2p-2)) = \sum_{p=2}^{m} p = m(m+1)/2-1,\\ O &= \sum_{p=2}^{m}(2p-2) = 2W-2(m-1).\end{aligned}$$

Since $|\mathrm{ov}(s_p,s_{p-1})|=2p-3$,

$$\mathrm{OPT}\leq |\langle s_m,s_{m-1},\ldots,s_2\rangle| = |s_2| + \sum_{p=3}^{m}(|s_p|-(2p-3)) = 4+\sum_{p=3}^{m}(p+1) = W+m.$$

Consequently,

$$O-\mathrm{OPT}\geq 2W-2(m-1)-(W+m) = W-3m+2.$$

Since $W=\Theta(m^2)$, for every fixed $\gamma<1$ and all sufficiently large m, $O-\mathrm{OPT}>\gamma W$. $\square$

AI disclosure

We used ChatGPT 5.6-Sol Pro to explore different approaches to prove the greedy conjecture. While we were trying to formulate a self-reduction for SCS, the agent suggested a proof of the inequality in Theorem 3.1, it was not directly related to the discussed question, but the agent had found a simpler route to the statement we tried to prove. The initial proof was complicated, but working together we simplified it significantly. Then we used the agent to do mechanical work to construct the example in Theorem 3.3. We also used Codex 5.6-Sol Ultra to find all typos in the paper. We have written all the arguments by ourselves and assume full responsibility for the manuscript.

References

[AS94] Chris Armen and Clifford Stein. A $2\frac{3}{4}$-Approximation Algorithm for the Shortest Superstring Problem. Technical report, Dartmouth College, Hanover, NH, USA, 1994.

[AS95] Chris Armen and Clifford Stein. Improved length bounds for the shortest superstring problem (extended abstract). In *WADS*, volume 955 of *Lecture Notes in Computer Science*, pages 494–505. Springer, 1995.

[AS96] Chris Armen and Clifford Stein. A 2 2/3-approximation algorithm for the shortest superstring problem. In *CPM*, volume 1075 of *Lecture Notes in Computer Science*, pages 87–101. Springer, 1996.

[BJJ97] Dany Breslauer, Tao Jiang, and Zhigen Jiang. Rotations of Periodic Strings and Short Superstrings. *Journal of Algorithms*, 24(2):340–353, 1997.

[BJL+91] Avrim Blum, Tao Jiang, Ming Li, John Tromp, and Mihalis Yannakakis. Linear approximation of shortest superstrings. In *STOC*, pages 328–336. ACM, 1991.

[CGPR94] Artur Czumaj, Leszek Gasieniec, Marek Piotrów, and Wojciech Rytter. Parallel and sequential approximations of shortest superstrings. In *SWAT*, volume 824 of *Lecture Notes in Computer Science*, pages 95–106. Springer, 1994.

[DB46] Nicolaas Govert De Bruijn. A combinatorial problem. In *Nederl. Akad. Wetensh. Proc.*, volume 49, pages 768–764, 1946.

[EMV22] Matthias Englert, Nicolaos Matsakis, and Pavel Veselý. Improved approximation guarantees for shortest superstrings using cycle classification by overlap to length ratios. In *STOC*, pages 317–330. ACM, 2022.

[EMV23] Matthias Englert, Nicolaos Matsakis, and Pavel Veselý. Approximation Guarantees for Shortest Superstrings: Simpler and Better. In *ISAAC*, volume 283 of *LIPIcs*, pages 29:1–29:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

[GMS80] John Gallant, David Maier, and James A. Storer. On Finding Minimal Length Superstrings. *Journal of Computer and System Sciences*, 20(1):50–58, 1980.

[GP14] Theodoros P. Gevezes and Leonidas S. Pitsoulis. The Shortest Superstring Problem. In *Optimization in Science and Engineering*, pages 189–227. Springer, 2014.

[KLSS03] Haim Kaplan, Moshe Lewenstein, Nira Shafrir, and Maxim Sviridenko. Approximation algorithms for asymmetric TSP by decomposing directed regular multigraphs. In *FOCS*, pages 56–65. IEEE Computer Society, 2003.

[KPS94] S Rao Kosaraju, James K Park, and Clifford Stein. Long tours and short superstrings. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 166–177. IEEE, 1994.

[KS05] Haim Kaplan and Nira Shafrir. The Greedy Algorithm for Shortest Superstrings. *Information Processing Letters*, 93(1):13–17, 2005.

[Lyn54] Roger C Lyndon. On burnside's problem. *Transactions of the American Mathematical Society*, 77(2):202–215, 1954.

[Ma09] Bin Ma. Why Greed Works for Shortest Common Superstring Problem. *Theoretical Computer Science*, 410(51):5374–5381, 2009.

[Muc07] Marcin Mucha. A tutorial on shortest superstring approximation. 2007.

[Muc13] Marcin Mucha. Lyndon Words and Short Superstrings. In *SODA*, pages 958–972. SIAM, 2013.

[PEvZ12] Katarzyna E. Paluch, Khaled M. Elbassioni, and Anke van Zuylen. Simpler Approximation of the Maximum Asymmetric Traveling Salesman Problem. In *STACS*, pages 501–506, 2012.

[Rau82] Gérard Rauzy. Suites à termes dans un alphabet fini. *Séminaire de théorie des nombres de Bordeaux*, pages 1–16, 1982.

[RBT04] Heidi J. Romero, Carlos A. Brizuela, and Andrei Tchernykh. An Experimental Comparison of Approximation Algorithms for the Shortest Common Superstring Problem. In *Mexican International Conference on Computer Science*, pages 27–34. IEEE Computer Society, 2004.

[Shi53] Anatolii Illarionovich Shirshov. Subalgebras of free lie algebras. *Matematicheskii Sbornik*, 75(2):441–452, 1953.

[Sto88] James A. Storer. *Data Compression: Methods and Theory*. Computer Science Press, 1988.

[SVB23] Ondřej Sladký, Pavel Veselý, and Karel Břinda. Masked Superstrings as a Unified Framework for Textual k-mer Set Representations. *bioRxiv*, 2023.

[Swe99] Z. Sweedyk. A $2\frac{1}{2}$-Approximation Algorithm for Shortest Superstring. *SIAM Journal on Computing*, 29(3):954–986, 1999.

[TU88] Jorma Tarhio and Esko Ukkonen. A Greedy Approximation Algorithm for Constructing Shortest Common Superstrings. *Theoretical Computer Science*, 57:131–145, 1988.

[Tur89] Jonathan S. Turner. Approximation Algorithms for the Shortest Common Superstring Problem. *Information and Computation*, 83(1):1–20, 1989.

[TY93] Shang-Hua Teng and F. Frances Yao. Approximating shortest superstrings. In *FOCS*, pages 158–165. IEEE Computer Society, 1993.

[Ukk90] Esko Ukkonen. A Linear-Time Algorithm for Finding Approximate Shortest Common Superstrings. *Algorithmica*, 5(3):313–323, 1990.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il