1991

A. Blum, T. Jiang, M. Li, J.T. Tromp, M. Yannakakis

Linear approximation of shortest superstrings

Computer Science/Department of Algorithmics and Architecture Report CS-R9126 April

CWI, nationaal instituut voor onderzoek op het gebied van wiskunde en informatica

CWI is the research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a non-profit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands organization for scientific research (NWO).

Linear Approximation of Shortest Superstrings

Avrim Blum* MIT
Tao Jiang† McMaster
Ming Li‡ Waterloo
John Tromp§ CWI
Mihalis Yannakakis¶ Bell Labs

We consider the following problem: given a collection of strings $s_1, \ldots, s_m$, find the shortest string s such that each s_i appears as a substring (a consecutive block) of s. Although this problem is known to be NP-hard, a simple greedy procedure appears to do quite well and is routinely used in DNA sequencing and data compression practice, namely: repeatedly merge the pair of strings with maximum overlap until only one string remains. Let n denote the length of the optimal superstring. A common conjecture states that the above greedy procedure produces a superstring of length $O(n)$ (in fact, $2n$), yet the only previous nontrivial bound known for any polynomial-time algorithm is a recent $O(n \log n)$ result.

We show that the greedy algorithm does in fact achieve a constant factor approximation, proving an upper bound of $4n$. Furthermore, we present a simple modified version of the greedy algorithm that we show produces a superstring of length at most $3n$. We also show the superstring problem to be *MAX SNP-hard*, which implies that a polynomial-time approximation scheme for this problem is unlikely.

1980 Mathematics Subject Classification: 69F13, 69F21, 69G12, 69L30.
CR Categories: F.1.3, F.2.2, G.2.2, J.3.
Keywords and Phrases: shortest common superstring, substring, approximation, learning, DNA sequencing, NP-hardness, greedy.
Note: This paper will appear in the proceedings of the 1991 ACM Symposium on Theory of Computing (STOC).

---

*Supported in part by an NSF Graduate Fellowship. Part of this work was done while the author was visiting at AT&T Bell Labs. Address: Lab for Computer Science, MIT, 545 Technology Sq., Cambridge, MA 02139. E-mail: avrim@theory.lcs.mit.edu

†Supported in part by a grant from SERB, McMaster University, and NSERC Operating Grant OGP0046613. Address: Department of Computer Science, McMaster University, Hamilton, Ont. L8S 4K1, Canada. E-mail: jiang@maccs.mcmaster.ca

‡Supported in part by the NSERC Operating Grants OGP0036747 and OGP0046506. Address: Department of Computer Science, University of Waterloo, Waterloo, Ont. N3L 3G1, Canada. E-mail: mli@water.waterloo.edu

§This work was supported in part by NSERC Grant OGP0036747 while the author was visiting at the University of Waterloo. Address: CWI, P.O. Box 4079, 1009 AB Amsterdam, The Netherlands. E-mail: tromp@cwi.nl

¶Address: Room 2C-319, AT&T Bell Labs, 600 Mountain Ave, Murray Hill, NJ 07974. E-mail: mihalis@research.att.com

Report CS-R9126
CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1 Introduction

Given a finite set of strings, we would like to find their shortest common superstring. That is, we want the shortest possible string s such that every string in the set is a substring of s.

The question is NP-hard [2, 1]. Due to its important applications in data compression [9] and DNA sequencing [8, 3, 4], efficient approximation algorithms for this problem are indispensable. We give an example from the DNA sequencing practice. A DNA molecule can be represented as a character string over the set of nucleotides $\{A, C, G, T\}$. Such a character string ranges from a few thousand symbols long for a simple virus to 3×10^9 symbols for a human being. Determining this string for different molecules, or *sequencing* the molecules, is a crucial step towards understanding the biological functions of the molecules. With current laboratory methods, only small fragments (chosen from unknown locations) of at most 500 base pairs can be sequenced at a time. Then from hundreds, thousands, sometimes millions of these fragments, a biochemist *assembles* the superstring representing the whole molecule. A simple greedy algorithm is routinely used [8, 3] to cope with this job. This algorithm, which we call GREEDY, just repeatedly merges the pair of strings with maximum overlap until only one string remains. It has been an open question as to how well GREEDY approximates a shortest common superstring, although a common conjecture states that GREEDY produces a superstring of length at most two times optimal [9, 10, 11].

From a different point of view, Li [4] considered learning a superstring from randomly drawn substrings in the Valiant learning model [12]. In a restricted sense, the shorter the superstring we obtain, the smaller the number of samples are needed to infer a superstring. Therefore finding a good approximation bound for shortest common superstring implies efficient learnability or inferability of DNA sequences [4]. Our linear approximation result reduces the number of samples required in [4] to learn a superstring by a multiplicative logarithmic factor.

Tarhio and Ukkonen [10] and Turner [11] established some performance guarantees for GREEDY with respect to so-called "compression" measure. This basically measures the number of symbols saved by GREEDY compared to plainly concatenating all the strings. It was shown that if the optimal solution saves m symbols, then GREEDY saves at least $m/2$ symbols. But, in general this implies no performance guarantee with respect to optimal length since in the best case this only says that GREEDY produces a superstring of length at most half the total length of all the strings.

Recently Li [4] obtained the first nontrivial bound for the superstring problem with respect to the optimal superstring length. Given a set of strings S, the algorithm in [4] produces a superstring of length at most $n \log n$ where n is the length of the shortest superstring for S. However, $n \log n$ is far from the conjectured $O(n)$ bound.

In this paper we show that the superstring problem *can* be approximated within a constant factor, and in fact that algorithm GREEDY produces a superstring of length at most $4n$. Furthermore, we give a simple modified greedy procedure MGREEDY that achieves a bound of $4n$, and then present another algorithm TGREEDY, based on MGREEDY, that we show achieves $3n$.

The rest of the paper is organized as follows: Section 2 contains notation, definitions, and some basic facts about strings. In Section 3 we describe our main algorithm MGREEDY with its proof. This proof forms the basis of the analysis in the next two sections. MGREEDY is improved to TGREEDY in Section 4. We finally give the $4n$ bound for GREEDY in Section 5. In Section 6, we show that the superstring problem is *MAX SNP-hard* which implies that it is unlikely there is a polynomial time approximation scheme for the superstring problem.

2 Preliminaries

Let $S = \{s_1, \ldots, s_m\}$ be a set of m strings over some alphabet Σ. Without loss of generality, we assume that set S is "substring free" in that no string $s_i \in S$ is a substring of any other $s_j \in S$. A *common superstring* of S is a string s such that each s_i in S is a substring of s. That is, for each s_i, string s can be written as $u_i s_i v_i$ for some u_i and v_i. In this paper, we will use n and OPT(S) interchangeably

for the length of the *shortest* common superstring for S. Our goal is to find a superstring for S whose length is as close to OPT(S) as possible.

For two strings s and t, not necessarily distinct, let v be the longest string such that $s = uv$ and $t = vw$ for some *non-empty* strings u and w. We call $|v|$ the (amount of) *overlap* between s and t, and denote it as $ov(s,t)$. Furthermore, u is called the *prefix* of s with respect to t, and is denoted $pref(s,t)$. Finally, we call $|pref(s,t)| = |u|$ the *distance* from s to t, and denote it as $d(s,t)$. So, the string $uvw = pref(s,t)t$, of length $d(s,t) + |t| = |s| + |t| - ov(s,t)$ is the shortest superstring of s and t in which s appears (strictly) before t, and is also called the *merge* of s and t. For $s_i, s_j \in S$, we will abbreviate $pref(s_i, s_j)$ to simply $pref(i,j)$, and $d(s_i, s_j)$ and $ov(s_i, s_j)$ to $d(i,j)$ and $ov(i,j)$ respectively. As an example of *self-overlap*, we have for the string $s =$ undergrounder, that $ov(s,s) = 5$. Also, $pref(s,s) =$ undergro and $d(s,s) = 8$.

Given a listing of strings $s_{i_1}, s_{i_2}, \ldots, s_{i_r}$, we define the superstring $s = \langle s_{i_1}, \ldots, s_{i_r} \rangle$ to be the string $pref(i_1,i_2)pref(i_2,i_3)\cdots pref(i_{r-1},i_r)s_{i_r}$. That is, s is the shortest string such that $s_{i_1}, s_{i_2}, \ldots, s_{i_r}$ appear *in order* in that string. For a superstring of a substring free set, this order is well-defined, since substrings cannot 'start' or 'end' at the same position, and if substring s_j starts before s_k, then s_j must also end before s_k. Define $first(s) = s_{i_1}$ and $last(s) = s_{i_r}$. In each iteration of GREEDY the following invariant holds:

Claim 1 *For two distinct strings s and t in GREEDY's set of strings, neither $first(s)$ nor $last(s)$ is a substring of t.*

Proof. Initially, $first(s) = last(s) = s$ for all strings, so the claim follows from the fact that S is substring free. The invariant could only be invalidated by a merge of two strings t_1 and t_2 into a string t that has, say, $first(s)$ as a substring. But if $first(s)$ was not yet a substring of either t_1 or t_2, then it must 'contain' the piece of overlap between t_1 and t_2, hence the overlap from t_1 to s would have been greater than the chosen one; a contradiction. □

So when GREEDY (or its variation MGREEDY that we introduce later) chooses s and t as having the maximum overlap, then the merge of s and t is $\langle first(s), \ldots, last(s), first(t), \ldots, last(t) \rangle$. s's order of substrings, wasn't the maximal one. This shows that $ov(s,t)$ equals $ov(last(s), first(t))$. For a permutation π on the set $\{1, \ldots, m\}$, let $S_\pi = \langle s_{\pi(1)}, \ldots, s_{\pi(m)} \rangle$. In a shortest superstring for S, the substrings appear in some total order, say $s_{\pi(1)}, \ldots, s_{\pi(m)}$, hence it must equal S_π.

Call two strings s, t equivalent, $s \equiv t$, if they are cyclic shifts of each other, i.e. if there are strings u, v such that $s = uv$ and $t = vu$. Finally, let s^k denote the string consisting of k copies of s concatenated together.

We will consider a traveling salesman problem on a weighted directed complete graph G_S derived from S and show that one can achieve a factor of 4 approximation for TSP on that graph, yielding a factor of 4 approximation for the shortest-common-superstring problem. Graph $G_S = (V, E, d)$ has m vertices $V = \{1, \ldots, m\}$, m^2 edges $E = \{(i,j) : 1 \leq i, j \leq m\}$. Here we take as weight function the distance $d(,)$: edge (i,j) has weight $d(i,j) = d(s_i, s_j)$, to obtain the *distance graph*. This graph is similar to one considered by Turner in the end of his paper [11]. Later we will take the overlap $ov(,)$ as the weight function to obtain the *overlap graph*. We will call s_i the string *associated* with vertex i, and let $pref(i,j) = pref(s_i, s_j)$ be the string associated to edge (i,j).

Notice now that TSP(G_S) $\leq$ OPT(S) $- ov(last(s), first(s)) \leq$ OPT(S), where TSP(G_S) is the cost of the minimum weight Hamiltonian cycle on G_S. The reason is that turning any superstring into a Hamiltonian cycle by overlapping its last and first substring saves on cost by charging $last(s)$ for only $d(last(s), first(s))$ instead of its full length.

We now define some notation for dealing with directed cycles in G_S. If c is a directed cycle in G_S with vertices $i_1, \ldots, i_r$ in order around c, we define $periods(c)$ to be the equivalence class $[pref(i_1,i_2)pref(i_2,i_3)\cdots pref(i_r,i_1)]$. A trailing index $[k]$ refers to the rotation starting with $pref(k, k+1)$. To avoid confusion with length, we will use $card(B)$ for the size of an equivalence class B. A moment's reflection shows that $card(B)$ is the minimum integer with the property that every member x of B is a power of some string y of length $card(B)$ (i.e., $x = y^k$ for some k). In general,

we will denote a cycle c with vertices $i_1, \ldots, i_r$ in order by "$i_1 \to \cdots \to i_r \to i_1$." Also, let $w(c)$, the *weight* of cycle c, equal $|s|, s \in periods(c)$. For convenience, we will say that s_j is in c, or "$s_j \in c$" if j is a vertex of the cycle c.

Now, a few preliminary facts about cycles in G_S. Let $c = i_0 \to \cdots \to i_{r-1} \to i_0$ and c' be cycles in G_S.

Claim 2 *Each string s_{i_j} in c is a substring of s^k for all $s \in periods(c)$ and sufficiently large k.*

Proof. By induction, s_{i_j} is a prefix of $pref(i_j, i_{j+1}) \cdots pref(i_{j+l-1}, i_{j+l})\, s_{i_{j+l}}$ for any $l \geq 0$ (addition modulo r). Every r consecutive prefixes form one 'loop' around the cycle: a string in $periods(c)$. Hence every substring of s_{i_j} of length $w(c)$ is in $periods(c)$, so take $k \geq |s_{i_j}|/w(c) + 1$. □

Claim 3 *If each of a substring free set of strings $\{s_1, \ldots, s_j\}$ is a substring of s^k for sufficiently large k, then there exists a cycle of weight $|s|$ containing all those strings.*

Proof. In a (infinite) repetition of s, every string s_i appears as a substring at every other $|s|$ characters. This naturally defines a circular ordering of the strings whose successive distances sum to $|s|$. □

Claim 4 *The superstring $\langle s_{i_0}, \cdots, s_{i_{r-1}} \rangle$ is a substring of $periods(c)[i_0]s_{i_0}$.*

Proof. String $\langle s_{i_0}, \ldots, s_{i_{r-1}} \rangle$ is clearly a substring of $\langle s_{i_0}, \ldots, s_{i_{r-1}}, s_{i_1} \rangle$, which by definition equals $pref(i_0, i_1) \cdots pref(i_{r-1}, i_0)s_{i_0} = periods(c)[i_0]s_{i_0}$. □

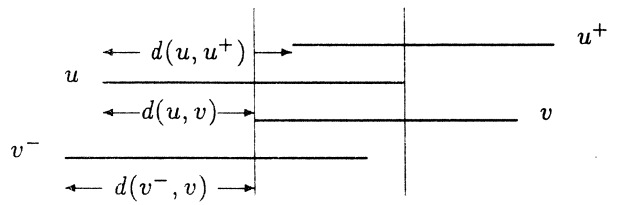
Claim 5 *If $periods(c') = periods(c)$, then there exists a third cycle $\tilde{c}$ with weight $w(c)$ containing all vertices in c and all those in c'.*

Proof. Follows from claims 2 and 3. □

Claim 6 *There exists a cycle $\tilde{c}$ of weight $card(periods(c))$ containing all vertices in c.*

Proof. Fix an $s \in periods(c)$ and let u be the shortest nonempty prefix of s such that $s = uv = vu$. Then $k = |u|$ divides $w(c)$, and $s = u^j$ where $j = w(c)/k$. Since k is the minimal (positive) number of rotations that takes s back to itself, we have $card(periods(c)) = k$. Now use Claim 3 for u. □

The following lemma has been proved in [10, 11]. In the picture below, the vertical bars surround pieces of string that match, showing a possible overlap between v^- and u^+, giving an upper bound on $d(v^-, u^+)$.



Lemma 7 *Let u, u^+, v^-, v be strings, not necessarily different, such that $ov(u, v) \geq \max\{ov(u, u^+), ov(v^-, v)\}$. Then, $ov(u, v) + ov(v^-, u^+) \geq ov(u, u^+) + ov(v^-, v)$, and $d(u, v) + d(v^-, u^+) \leq d(u, u^+) + d(v^-, v)$.*

That is, given the choice of merging u to u^+ and v^- to v or instead merging u to v and v^- to u^+, the best choice is that which contains the pair of largest overlap.

3 A $4 \cdot \mathrm{OPT}(S)$ bound for a modified greedy algorithm

Let S be a set of strings and G_S the associated graph. Now, although finding a minimum weight Hamiltonian cycle in a weighted directed graph is in general a hard problem, there *is* a polynomial-time algorithm for a similar problem known as the *assignment problem* [5]. Here, the goal is simply to find a decomposition of the graph into cycles such that each vertex is in exactly one cycle and the total weight of the cycles is minimized. Let $\mathrm{CYC}(G_S)$ be the weight of the minimum assignment on graph G_S, so $\mathrm{CYC}(G_S) \leq \mathrm{TSP}(G_S) \leq \mathrm{OPT}(S)$.

The proof that a modified greedy algorithm MGREEDY finds a superstring of length at most $4 \cdot \mathrm{OPT}(S)$ proceeds in two stages. We first show that an algorithm that finds an optimal assignment on G_S, then opens each cycle into a single string, and finally concatenates all such strings together has a performance ratio of at most 4. We then show (Theorem 10) that in fact for the particular graphs G_S, a greedy strategy can be used to find optimal assignments.

Consider the following algorithm for finding a superstring of the strings in S.

Algorithm Concat-Cycles

1. On input S, create graph G_S and find a minimum weight assignment C on G_S. Let C be the collection of cycles $\{c_1, \ldots, c_p\}$.
2. For each cycle $c_i = i_1 \rightarrow \cdots \rightarrow i_r \rightarrow i_1$, let $\tilde{s}_i = \langle s_{i_1}, \ldots, s_{i_r} \rangle$ be the string obtained by opening c_i, where i_1 is arbitrarily chosen. The string $\tilde{s}_i$ has length at most $w(c_i) + |s_{i_1}|$ by Claim 4.
3. Concatenate together the strings $\tilde{s}_i$ and produce the resulting string $\tilde{s}$ as output.

Theorem 8 *Algorithm Concat-Cycles produces a string of length at most* $4 \cdot OPT(S)$.

Before proving Theorem 8, we first need a preliminary lemma giving an upper bound on the amount of overlap possible between strings in two different cycles of C.

Lemma 9 *Let c and c' be two cycles in a minimum weight assignment C with $s \in c$ and $s' \in c'$. Then, the overlap between s and s' is less than $w(c) + w(c')$.*

Proof. Let $x = \mathit{periods}(c)$ and $x' = \mathit{periods}(c')$. Since C is a minimum weight assignment, we know by Claim 5 that $x \neq x'$. In addition, by Claim 6, $\mathit{card}(x) = w(c)$ and $\mathit{card}(x') = w(c')$.

Suppose for contradiction that s and s' overlap in a string u with $|u| \geq w(c) + w(c')$. We now consider two cases. First, suppose $w(c) = w(c')$. In this case, s and s' have in common some substring y of length $|y| = w(c) = w(c')$. So by Claim 2, $x = [y] = x'$, which as noted above contradicts the minimality of C.

The second case is that $w(c) \neq w(c')$, and without loss of generality assume that $w(c) > w(c')$. Denoting the substring of u starting at the i-th symbol and ending at the j-th as $u_{i,j}$, we have $u_{1,w(c)} = u_{1+w(c'),w(c)+w(c')} = u_{1,w(c')}u_{1+w(c'),w(c)}$. This shows that $\mathit{periods}(c)$ has less than $w(c)$ different rotations, which by Claim 6 contradicts the minimality of C. □

Proof of Theorem 8. Since $C = \{c_1, \ldots, c_p\}$ is an optimal assignment, $\mathrm{CYC}(G_S) = \sum_{i=1}^{p} w(c_i) \leq \mathrm{OPT}(S)$. A second lower bound on $\mathrm{OPT}(S)$ can be determined as follows: For each cycle c_i, let $w_i = w(c_i)$ and l_i denote the length of the longest string in c_i. By Lemma 9, if we consider the longest string in each cycle and merge them together optimally, the total amount of overlap will be at most $2\sum_{i=1}^{p} w_i$. So the resulting string will have length at least $\sum_{i=1}^{p} l_i - 2w_i$. Thus $\mathrm{OPT}(S) \geq \max(\sum_{i=1}^{p} w_i, \sum_{i=1}^{p} l_i - 2w_i)$.

The output string $\tilde{s}$ of algorithm Concat-Cycles has length at most $\sum_{i=1}^{p} l_i + w_i$ (Claim 4). So,

$$|\tilde{s}| \leq \sum_{i=1}^{p} l_i + w_i$$

$$\begin{aligned} &= \sum_{i=1}^{p} l_i - 2w_i + \sum_{i=1}^{p} 3w_i \\ &\leq \mathrm{OPT}(S) + 3 \cdot \mathrm{OPT}(S) \\ &= 4 \cdot \mathrm{OPT}(S). \quad \square \end{aligned}$$

We are now ready to present the algorithm MGREEDY, and show that in fact it mimics algorithm Concat-Cycles.

Algorithm MGREEDY

1. Let S be the input set of strings and T be empty.
2. While S is non-empty, do the following: Choose $s, t \in S$ (not necessarily distinct) such that $ov(s,t)$ is maximized, breaking ties arbitrarily. If $s \neq t$, then remove s and t from S and replace them with the merged string $\langle s,t \rangle$. If $s = t$, then just remove s from S and add it to T.
3. When S is empty, output the concatenation of the strings in T.

We can look at MGREEDY as choosing edges in the overlap graph $(V = S, E = V \times V, ov(,))$. When MGREEDY chooses strings s and t as having the maximum overlap (where t may equal s), it chooses the directed edge from $last(s)$ to $first(t)$ (see Claim 1). Thus, MGREEDY constructs/joins paths, and closes them into cycles, to end up with a collection of disjoint cycles $M \subset E$ that cover the vertices of G_S. We will call M the assignment created by MGREEDY. Now think of MGREEDY as taking a list of all the edges sorted by overlap (resolving ties in some definite way), and going down the list deciding for each edge whether to include it or not. Let us say that an edge e *dominates* another edge f if e precedes f in this list and shares its head or tail with f. MGREEDY includes an edge f if and only if it has not yet included a dominating edge.

Theorem 10 *The assignment created by algorithm MGREEDY is an optimal assignment.*

Proof. Note that the overlap weight of an assignment and its distance weight add up to the total length of all strings. Accordingly, an assignment is optimal (i.e., has minimum total weight in the distance graph) if and only if it has maximum total overlap. Among the maximum overlap assignments, let N be one that has the maximum number of edges in common with M. We shall show that $M = N$.

Suppose this is not the case, and let e be the edge of maximum overlap in the symmetric difference of M and N, with ties broken the same way as by MGREEDY. Suppose first that this edge is in $N \setminus M$. Since MGREEDY did not include e, it must have included another adjacent edge f that dominates e. Edge f cannot be in N (since N is an assignment), therefore f is in $M \setminus N$, contradicting our choice of the edge e. Suppose that $e = k \rightarrow j$ is in $M \setminus N$. The two N edges $i \rightarrow j$ and $k \rightarrow l$ that share head and tail with e are not in M, and thus are dominated by e. Replacing in N these two edges with $e = k \rightarrow j$ and $i \rightarrow l$ would yield an assignment N' that has more edges in common with M and, by Lemma 7, has no less overlap than N. This would contradict our choice of N. $\square$

Since algorithm MGREEDY finds an optimal assignment, the string it produces is no longer than the string produced by algorithm Concat-Cycles. (In fact, it could be shorter since it breaks each cycle in the optimum position.)

4 Improving to $3 \cdot \mathrm{OPT}(S)$

Recall that in the last step of algorithm MGREEDY, we simply concatenate all the strings in set T without any compression. Intuitively, if we instead try to overlap the strings in T, we might be able to achieve a bound better than $4 \cdot \mathrm{OPT}(S)$. Let TGREEDY denote the algorithm that operates in the

same way as MGREEDY except that in the last step, it merges the strings in T by running GREEDY on them. We can show that TGREEDY indeed achieves a better bound: it produces a superstring of length at most $3 \cdot \mathrm{OPT}(S)$.

Theorem 11 *Algorithm TGREEDY produces a superstring of length at most* $3 \cdot OPT(S)$.

Proof. Let S be a set of strings and s be the superstring obtained by TGREEDY on S. Let $n = \mathrm{OPT}(S)$ be the length of a shortest superstring of S. We show that $|s| \leq 3n$.

Let T be the set of all "self-overlapping" strings obtained by MGREEDY on S and C be the assignment created by MGREEDY. For each $x \in T$, let c_x denote the cycle in C corresponding to string x, and let $w_x = w(c_x)$ be its weight. Denote $\sum_{s \in S} |s|$, the total length of all strings in a set S by $||S||$, and define $w = \sum_{x \in T} w_x$. From the proof of Theorem 8, we have that $w \leq n$.

For each $x \in T$, let x_0 be an arbitrary string in c_x. Let $S' = \{x_0 | x \in T\}$, $n' = \mathrm{OPT}(S')$, $S'' = \{periods(c_x)[0]x_0 | x \in T\}$, and $n'' = \mathrm{OPT}(S'')$.

By Claim 4, a superstring for S'' is also a superstring for T, so $n_T \leq n''$, where $n_T = \mathrm{OPT}(T)$. For any permutation π on T elements, we have $|S''_\pi| \leq |S'_\pi| + \sum_{x \in T} w_x$, so $n'' \leq n' + w$. Observe that $S' \subseteq S$ implies $n' \leq n$. Summing up, we get

$$n_T \leq n'' \leq n' + w \leq n + w.$$

By Lemma 9, the compression achieved in a shortest superstring of T is less than $2w$, i.e., $||T|| - n_T \leq 2w$. By the results in [10, 11], we know that the compression achieved by GREEDY on set T is at least half the compression achieved in any superstring of T. That is, $||T|| - |s| \geq (||T|| - n_T)/2 = ||T|| - n_T - (||T|| - n_T)/2 \geq ||T|| - n_T - w$. So, $|s| \leq n_T + w$. Combined with $n_T \leq n + w$, this gives $|s| \leq n + 2w \leq 3n$. □

5 GREEDY achieves linear approximation

One would expect that an analysis similar to that of MGREEDY would also work for the original GREEDY. This turns out not to be the case. The analysis of GREEDY is severely complicated by the fact that it continues processing the "self-overlapping" strings. MGREEDY was especially designed to avoid these complications, by separating such strings. Let $GREEDY(S)$ denote the length of the superstring produced by GREEDY on a set S. It is tempting to claim that

$$GREEDY(S \cup \{s\}) \leq GREEDY(S) + |s|.$$

If this were true, a simple argument would extend the $4 \cdot \mathrm{OPT}(S)$ result for MGREEDY to GREEDY. But the following counterexample disproves this seemingly innocent claim. Let

$$S = \{ca^m, a^{m+1}c^m, c^m b^{m+1}, b^m c\}, s = b^{m+1}a^{m+1}.$$

Now $GREEDY(S) = |ca^{m+1}c^m b^{m+1}c| = 3m+4$, whereas $GREEDY(S \cup \{s\}) = |b^m c^m b^{m+1} a^{m+1} c^m a^m| = 6m + 2 > (3m + 4) + (2m + 2)$.

With a more complicated analysis we will nevertheless show that

Theorem 12 *GREEDY produces a string of length at most* $4 \cdot OPT(S)$.

Proof. Think of both GREEDY and MGREEDY as taking a list of all edges sorted by overlap, and going down the list deciding for each edge whether to include it or not. Call an edge *better* (*worse*) if it appears before (after) another in this list. Better edges have at least the overlap of worse ones. Recall that an edge dominates another iff it is better and shares its head or tail with the other one.

At the end, GREEDY has formed a Hamiltonian path

$$s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_m$$

of 'greedy' edges. For convenience we will usually abbreviate s_i to i. Greedy does not include an edge f iff

1. f is dominated by an already chosen edge e, or

2. f is not dominated but it would form a cycle.

Let us call the latter "bad back edges"; a bad back edge $f = j \rightarrow i$ necessarily has $i \leq j$. Each bad back edge $f = j \rightarrow i$ corresponds to a string $\langle s_i, s_{i+1}, \ldots, s_j \rangle$ that, at some point in the execution of Greedy, has more (self) overlap than the pair that is merged. When GREEDY considers f, it has already chosen all (better) edges on the greedy path from i to j, but not yet the (worse) edges $i-1 \rightarrow i$ and $j \rightarrow j+1$ (if they exist). Bad back edge f is said to *span* the closed interval $I_f = [i, j]$. The above observations imply easily the following lemma.

Lemma 13 *Let e and f be two bad back edges. The closed intervals I_e and I_f are either disjoint, or one contains the other. If $I_e \supset I_f$ then e is worse than f (thus, $ov(e) \leq ov(f)$).*

Thus, the intervals of the bad back edges are nested. *Culprits* are the minimal (innermost) such intervals. Each culprit $[i, j]$ corresponds to a *culprit string* $\langle s_i, s_{i+1}, \ldots, s_j \rangle$. Note that if $f = j \rightarrow i$ is the back edge of a culprit $[i, j]$, and e is another bad back edge that shares head or tail with f, then $I_e \supset I_f$, and therefore f dominates e.

Call the worst edge between every two successive culprits on the greedy path a *weak link*. Note that weak links are also worse than all edges in the two adjacent culprits as well as their back edges. If we remove all the weak links, the greedy path is partitioned into a set of paths, called *blocks*. Every block consists of a nonempty culprit as the middle segment, and (possibly empty) left and right *extensions*. The set of strings (nodes) S is thus partitioned into three sets S_l, S_m, S_r of left, middle, and right strings.

Lemma 14 *Let $f = j \rightarrow i$ be a bad back edge. Node i is either a left node or the first node of a culprit. Node j is either a right node or the last node of a culprit.*

Proof. Let $c = [k, l]$ be the leftmost culprit in I_f. Now either $i = k$ is the first node of c, or $i < k$ is in the left extension of c, or $i < k$ is in the right extension of the culprit c' to the left of c. In the latter case however, I_f includes the weak link, which by definition is worse than all edges between the culprits c' and c, including the edge $i-1 \rightarrow i$. This contradicts the observation preceding Lemma 13. A similar argument holds for s_j. □

Let C_m be the assignment on the set S_m of middle strings (nodes) that has one cycle for each culprit, consisting of the greedy edges together with the back edge of the culprit. If we consider the application of the algorithm MGREEDY on the subset of strings S_m, it is easy to see that the algorithm will actually construct the assignment C_m. Theorem 10 implies then the following lemma.

Lemma 15 *C_m is an optimal assignment on the set S_m of middle strings.*

Let the graph $G_l = (V_l, E_l)$ consist of the left/middle part of all blocks in the greedy path, i.e. $V_l = S_l \cup S_m$ and E_l is the set of non-weak greedy edges between nodes of V_l. Let M_l be a maximum overlap assignment on V_l, as created by MGREEDY on the ordered sublist of edges in $V_l \times V_l$. Let $V_r = S_m \cup S_r$, and define similarly the graph $G_r = (V_r, E_r)$ and the optimal assignment M_r on the right/middle strings. Let l_c be the sum of the lengths of all culprit strings. Define $l_l = \sum_{i \in S_l} d(s_i, s_{i+1})$ as the total length of all left extensions and $l_r = \sum_{i \in S_r} d(s_i^R, s_{i-1}^R)$ as the total length of all right extensions. The length of the string produced by GREEDY is $l_l + l_c + l_r - o_w$, where o_w is the summed block overlap (i.e. the sum of the overlaps of the weak links).

Denoting the overlap $\sum_{e \in E} ov(e)$ of a set of edges E as $ov(E)$, define the cost of a set of edges E on a set of strings (nodes) V as

$$cost(E) = ||V|| - ov(E).$$

Note that the distance plus overlap of a string s to another equals $|s|$. Because an assignment (e.g. M_l or M_r) has an edge from each node, its cost equals its distance weight. Since V_l and V_r are subsets

of S and M_l and M_r are optimal assignments, we have $cost(M_l) \leq n$ and $cost(M_r) \leq n$. For E_l and E_r we have that $cost(E_l) = l_l + l_c$ and $cost(E_r) = l_r + l_c$.

We have established the following (in)equalities:

$$\begin{aligned} & l_l + l_c + l_r \\ = \; & (l_l + l_c) + (l_c + l_r) - l_c \\ = \; & cost(E_l) + cost(E_r) - l_c \\ = \; & ||V_l|| - ov(E_l) + ||V_r|| - ov(E_r) - l_c \\ = \; & cost(M_l) + ov(M_l) - ov(E_l) + cost(M_r) + \\ & + ov(M_r) - ov(E_r) - l_c \\ \leq \; & 2n + ov(M_l) - ov(E_l) + ov(M_r) - ov(E_r) - l_c. \end{aligned}$$

We proceed by bounding the overlap differences in the above equation. We combine the left/middle and middle/right parts as follows.

Let V be the disjoint union of V_l and V_r, let E be the disjoint union of E_l and E_r, and let $G = (V, E)$ be the disjoint union of G_l and G_r. Thus each string in $S_l \cup S_r$ occurs once, while each string in S_m occurs twice in G. We modify E to take advantage of the block overlaps. Add each weak link to E as an edge from the last node in the corresponding middle/right path of G_r to the first node of the corresponding left/middle path of G_l. This procedure yields a new set of edges E'. Its overlap equals $ov(E') = ov(E_l) + ov(E_r) + o_w$.

Let M be the disjoint union of M_l and M_r, an assignment on V. Its overlap equals $ov(M) = ov(M_l) + ov(M_r)$. Every edge of M connects two V_l nodes or two V_r nodes; thus, all edges of M satisfy the hypothesis of the following lemma.

Lemma 16 *Let N be any assignment on V. Let $e = t \rightarrow h$ be an edge of $N \setminus E'$ that is not in $V_r \times V_l$. Then e is dominated by either*

1. *an adjacent E' edge, or*
2. *a culprit's back edge with which it shares the head h and $h \in V_r$, or*
3. *a culprit's back edge with which it shares the tail t and $t \in V_l$.*

Proof. Suppose first that e corresponds to a bad back edge. By Lemma 14, h corresponds to a left node or to the first node of a culprit. In the latter case, e is dominated by the back edge of the culprit (see the comment after Lemma 13). Therefore, either h is the first node of a culprit in V_r (and case 2 holds), or else $h \in V_l$. Similarly, either t is the last node of a culprit in V_l (and case 3 holds) or else $t \in V_r$. Since e is not in $V_r \times V_l$, it follows then that case 2 or case 3 holds.

Suppose that e does not correspond to a bad back edge. Then it must be dominated by some greedy edge f. If the corresponding greedy edge sharing head or tail with e is in E' then we have case 1. If it is not in E', then either h is the first node of a culprit in V_r or t is the last node of a culprit in V_l, and in both cases f is dominated by the back edge of the culprit. Thus, we have case 2 or 3. □

While Lemma 16 ensures that each edge of M is bounded in overlap, it may be that some edges of E' are double charged. We will modify M without decreasing its overlap and without invalidating Lemma 16 into an assignment M' such that each edge of E' is dominated by one of its adjacent M' edges.

Lemma 17 *Let N be any assignment on V such that $N \setminus E'$ does not contain any edges in $V_r \times V_l$. Then there is an assignment N' on V satisfying the following properties.*

1. *$N' \setminus E'$ has also no edges in $V_r \times V_l$,*
2. *$ov(N') \geq ov(N)$,*

3. each edge in $E' \setminus N'$ is dominated by one of its two adjacent N' edges.

Proof. It suffices to argue that if N violates property 3, then we can construct another assignment N' that satisfies properties 1 and 2, and which has more edges in common with E'.

Let $e = k \rightarrow j$ be an edge in $E' - N$ that dominates both adjacent N edges, $f = i \rightarrow j$, and $g = k \rightarrow l$. By Lemma 7, replacing edges f and g of N with e and $i \rightarrow l$ produces an assignment N' with at least as large overlap. To see that the new edge $i \rightarrow l$ of $N' \setminus E'$ is not in $V_r \times V_l$, observe that if $i \in V_r$ then $j \in V_r$ because of the edge $f = i \rightarrow j$ ($N \setminus E'$ does not have edges in $V_r \times V_l$), which implies that k is in V_r because of the E' edge $e = k \rightarrow j$ (E' does not have edges in $V_l \times V_r$), which implies that also $l \in V_r$ because of the N edge $g = k \rightarrow l$. □

By Lemmas 16 and 17, we can construct from the assignment M another assignment M' with at least as large total overlap, and such that we can charge the overlap of each edge of M' to an edge of E' or to the back edge of a culprit. Every edge of E' is charged for at most one edge of M', while the back edge of each culprit is charged for at most two edges of M': for the M' edge entering the first culprit node in V_r and the edge coming out of the last culprit node in V_l. Therefore, $ov(M) \leq ov(M') \leq ov(E') + 2o_c$, where o_c is the summed overlap of all culprit back edges. Denote by l_w the summed weight of all culprit cycles, i.e., the weight of the (optimal) assignment C_m on S_m from Lemma 15. As in the proof of MGREEDY, we have $l_c = l_w + o_c$, $o_c - 2l_w \leq n$, and $l_w \leq n$. Putting everything together, the string produced by GREEDY has length

$$
\begin{aligned}
& l_l + l_c + l_r - o_w \\
\leq\ & 2n + ov(M_l) - ov(E_l) + ov(M_r) - ov(E_r) - l_c - o_w \\
\leq\ & 2n + ov(M') - ov(E') - l_c \\
\leq\ & 2n + 2o_c - l_c \\
=\ & 2n + o_c - l_w \\
\leq\ & 3n + l_w \\
\leq\ & 4n.
\end{aligned}
$$

□

6 Lower bound

We show here that the superstring problem is *MAX SNP-hard*. This implies that if there is a polynomial time approximation scheme for the superstring problem, then there is one also for a wide class of optimization problems, including several variants of maximum satisfiability, the node cover and independent set problem in bounded-degree graphs, max cut, etc. This is considered rather unlikely.

Let A, B be two optimization (maximization or minimization) problems. We say that A *L-reduces* (for *linearly reduces*) to B if there are two polynomial time algorithms f and g and constants α and $\beta > 0$ such that:

1. Given an instance a of A, algorithm f produces an instance b of B such that the cost of the optimum solution of b, $opt(b)$, is at most $\alpha \cdot opt(a)$, and
2. Given any solution y of b, algorithm g produces in polynomial time a solution x of a such that $|cost(x) - opt(a)| \leq \beta |cost(y) - opt(b)|$.

Some basic facts about L-reductions are: First, the composition of two L-reductions is also an L-reduction. Second, if problem A L-reduces to problem B and B can be approximated in polynomial time with relative error ϵ (i.e., within a factor of $1+\epsilon$ or $1-\epsilon$ depending on whether B is a minimization or maximization problem) then A can be approximated with relative error $\alpha\beta\epsilon$. In particular, if B

has a polynomial time approximation scheme, then so does A. The class MAX SNP is a class of optimization problems defined syntactically in [6]. It is known that every problem in this class can be approximated within *some* constant factor. A problem is MAX SNP-hard if every problem in MAX SNP can be L-reduced to it.

Theorem 18 *The superstring problem is MAX SNP-hard.*

Proof. The reduction is from a special case of the TSP with triangle inequality. Let TSP(1,2) be the TSP restricted to instances where all the distances are either 1 or 2. We can consider an instance to this problem as being specified by a graph H; the edges of H are precisely those that have length 1 while the edges that are not in H have length 2. We need here the version of the TSP where we seek the shortest Hamiltonian path (instead of cycle), and, more importantly, we need the additional restriction that the graph H be of bounded degree (the precise bound is not important). It was shown in [7] that the TSP(1,2) problem (even for this restricted version) is MAX SNP-hard.

Let H be a graph of bounded degree D specifying an instance of TSP(1,2). The hardness result holds for both the symmetric and the asymmetric TSP (i.e., for both undirected and directed graphs H). We let H be a directed graph here. The reduction is similar to the one of [1] used to show the NP-completeness of the superstring decision problem. We have to prove here that it is an L-reduction. For every vertex v of H we have two letters v and v'. In addition there is one more letter $\#$. Corresponding to each vertex v we have a string $v\#v'$, called the *connector* for v. For each vertex v, enumerate the edges out of v in an arbitrary cyclic order as $(v, w_0), \ldots, (v, w_{d-1})$ (*). Corresponding to the ith edge (v, w_i) out of v we have a string $p_i(v) = v'w_{i-1}v'w_i$, where subscript arithmetic is modulo d. We will say that these strings are *associated* with v.

Let n be the number of vertices and m the number of edges of H. If all vertices have degree at most D then $m \leq Dn$. Let k be the minimum number of edges whose addition to H suffices to form a Hamiltonian path. Thus, the optimal cost of the TSP instance is $n - 1 + k$. We shall argue that the length of the shortest common superstring is $2m + 3n + k$. It will follow then that the reduction is linear since m is linear in n.

Consider the distance-weighted graph G_S for this set of strings, and let G_2 be its subgraph with only edges of minimal weight (2). Clearly, G_2 has exactly one component for each vertex of H, which consists of a cycle of the associated p strings, and a connector that has an edge to each of them. We need only consider 'standard' superstrings in which all strings associated with some vertex form a subgraph of G_2, so that only the last p string has an outgoing edge of weight more than 2 (3 or 4). Namely, if some vertex fails this requirement, then at least two of its associated strings have outgoing edges of weight more than 2, thus we do not increase the length by putting all its p strings directly after its connector in a standard way. A standard superstring naturally corresponds to an ordering of vertices $v_1, v_2, \ldots, v_n$.

For the converse there remains a choice of which string q succeeds a connector $v_i\#v_i'$. If H has an edge from v_i to v_{i+1} and the 'next' edge out of v_i (in (*)) goes to, say v_j, then choosing $q = v_i'v_{i+1}v_i'v_j$ results in a weight of 3 on the edge from the last p string to the next connector $v_{i+1}\#v_{i+1}'$, whereas this weight would otherwise be 4. If H doesn't have this edge, then the choice of q doesn't matter. Let us call a superstring 'Standard' if in addition to being standard, it also satisfies this latter requirement for all vertices.

Now suppose that the addition of k edges to H gives a Hamiltonian path $v_1, v_2, \ldots, v_{n-1}, v_n$. Then we can construct a corresponding Standard superstring. If the out-degree of v_i is d_i, then its length will be $\sum_{i=1}^{n}(2 + 2d_i + 1) + k + 3 = 3n + 2m + k$.

Conversely, suppose we are given a common superstring of length $3n + 2m + k$. This can then be turned into a Standard superstring that is no longer. If $v_1, v_2, ..., v_n$ is the corresponding order of vertices, it follows that H cannot be missing more than k of the edges (v_i, v_{i+1}). □

Since the strings in the above L-reduction have bounded length (4), the reduction applies also to the maximization version of the superstring problem [11, 10]. That is, maximizing the total compression is also MAX SNP-hard.

7 Open problems

We end the paper with several open questions raised from this research:

(1) Obtain an algorithm which achieves a better performance than $3 \cdot \mathrm{OPT}(S)$.

(2) Prove or disprove the conjecture that GREEDY achieves $2 \cdot \mathrm{OPT}(S)$.

(3) Consider the following problem: Given a set (of positive examples) S_p and a set (of negative examples) S_n, find a shortest superstring s such that s is a common superstring of S_p but no string in S_n is a substring of s. The Group-Merge algorithm in [4] achieves $O(\mathrm{OPT}(S) \cdot \log(\mathrm{OPT}(S))$ for this problem. Can this be improved to $O(\mathrm{OPT}(S))$? Notice that GREEDY and its variations do not work for this problem.

8 Acknowledgments

We thank Samir Khuller and Vijay Vazirani for discussions on the superstring algorithms (Samir brought the authors together).

References

[1] J. Gallant, D. Maier, J. Storer. On finding minimal length superstring. *Journal of Computer and System Sciences 20, 50-58, 1980.*

[2] M. Garey and D. Johnson. Computers and Intractability. *Freeman, New York, 1979.*

[3] A. Lesk (Edited). Computational Molecular Biology, Sources and Methods for Sequence Analysis. *Oxford University Press, 1988.*

[4] M. Li. Towards a DNA sequencing theory. *31st IEEE Symp. on Foundations of Computer Science, 125-134, 1990.*

[5] C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, 1982.

[6] C. Papadimitriou and M Yannakakis. Optimization, approximation and complexity classes. *20th ACM Symp. on Theory of Computing*, 229-234, 1988.

[7] C. Papadimitriou and M Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research.* To appear.

[8] H. Peltola, H. Soderlund, J. Tarhio, and E. Ukkonen. Algorithms for some string matching problems arising in molecular genetics. *Information Processing 83 (Proc. IFIP Congress, 1983) 53-64.*

[9] J. Storer. *Data compression: methods and theory. Computer Science Press, 1988.*

[10] J. Tarhio and E. Ukkonen. A Greedy approximation algorithm for constructing shortest common superstrings. *Theoretical Computer Science 57 131-145 1988*

[11] J. Turner. Approximation algorithms for the shortest common superstring problem. *Information and Computation 83 1-20 1989.*

[12] L. G. Valiant. A Theory of the Learnable. *Comm. ACM* 27(11) 1134-1142 1984.