Linear Approximation of Shortest Superstrings

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

Abstract

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 (distinct) 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.

Key words: Shortest Common Superstring, Approximation Algorithms.

*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. N2L 3G1, Canada. E-mail: mli@math.uwaterloo.ca

§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

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 [5, 6]. Due to its important applications in data compression [14] and DNA sequencing [8, 9, 13], 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 approximately 3×10^9 symbols for a human being. Determining this representation 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 bases 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, 13] to cope with this job. This algorithm, which we call GREEDY, repeatedly merges the pair of (distinct) 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 [14, 15, 16].

From a different point of view, Li [9] considered learning a superstring from randomly drawn substrings in the Valiant learning model [17]. 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 [9]. Our linear approximation result improves Li's $O(n \log n)$ approximation by a multiplicative logarithmic factor.

Tarhio and Ukkonen [15] and Turner [16] established some performance guarantees for GREEDY with respect to the "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 l symbols, then GREEDY saves at least $l/2$ 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.

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 also 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 7, we show that the superstring problem is *MAX SNP-hard* which implies that there is unlikely to exist a polynomial time approximation scheme for the superstring problem.

2 Preliminaries

Let $S = \{s_1, \ldots, s_m\}$ be a set of strings over some alphabet Σ. 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 $s_j \in S$. 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, the 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 $\mathrm{OPT}(S)$ 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 $\mathrm{OPT}(S)$ as possible.

Example. Assume we want to find the shortest common superstring of all words in the following sentence: "Alf ate half lethal alpha alfalfa". The word "alf" is a substring of both "half" and "alfalfa", so we can immediately eliminate it. Our set of words is now $S_0 = \{$ ate, half, lethal, alpha, alfalfa $\}$. A trivial superstring is "atehalflethalalphaalfalfa" of length 25, which is simply the concatenation of all substrings. A shortest common superstring is "lethalphalfalfate", of length 17, saving 8 characters over the previous one (a compression of 8). Looking at what GREEDY would make of this example, we see that it would start out with the largest overlaps from "lethal" to "half" to "alfalfa" producing "lethalfalfa". It then has 3 choices of single character overlap, two of which lead to another shortest superstring "lethalfalfalphate", and one of which is lethal in the sense of giving a superstring that is one character longer. In fact, it is easy to give an example where GREEDY outputs a string almost twice as long as the optimal one, for instance on input $\{c(ab)^k, (ba)^k, (ab)^k c\}$.

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. The overlap between a string and itself is called a *self-overlap*. As an example of self-overlap, we have for the string $s =$ undergrounder an overlap of $ov(s, s) = 5$ Also, $pref(s, s) =$ undergro and $d(s, s) = 8$. The string $s =$ alfalfa, for which $ov(s, s) = 4$, shows that the overlap is not limited to half the total string length .

Given a list 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. Suppose that the invariant is invalidated by a merge of two strings t_1 and t_2 into a string $t = \langle t_1, t_2 \rangle$ that has, say, $first(s)$ as a substring. Let $t = u\ first(s)\ v$. Since $first(s)$ is not a substring of either t_1 or t_2, it must properly 'contain' the piece of overlap between t_1 and t_2, *i.e.*, $|first(s)| > ov(t_1, t_2)$ and $|u| < d(t_1, t_2)$. Hence, $ov(t_1, s) > ov(t_1, t_2)$; a contradiction. ■

So when GREEDY (or its variation MGREEDY that we introduce later) chooses s and t as having the maximum overlap, then this overlap $ov(s, t)$ in fact equals $ov(last(s), first(t))$, and as a result, the merge of s and t is $\langle first(s), \ldots, last(s), first(t), \ldots, last(t) \rangle$. We can therefore say that GREEDY *orders* the substrings, by finding the shortest superstring in which the substrings appear in that order.

We can rephrase the above in terms of permutations. 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.

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 $V = \{1, \ldots, m\}$, and m^2 edges $E = \{(i, j) : 1 \leq i, j \leq m\}$. Here we take as weight

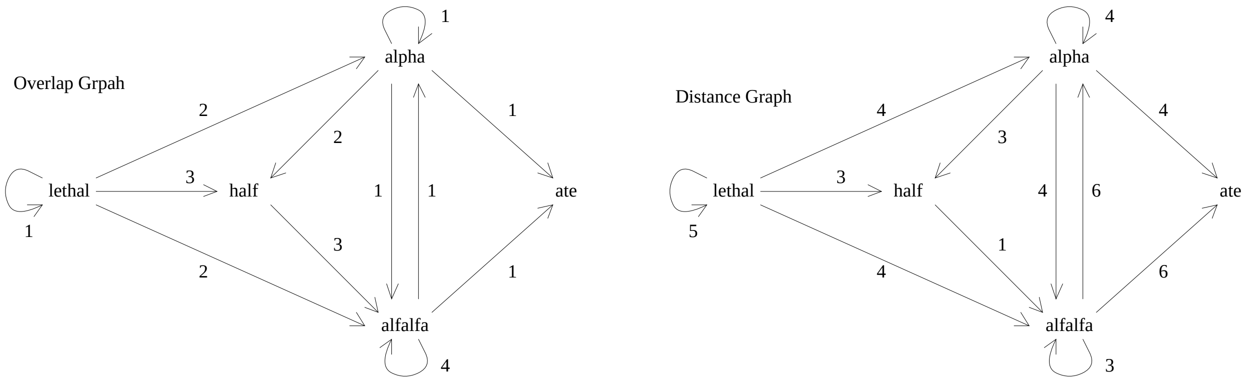

Figure 1: The overlap and distance graphs.

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 [16]. 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 $\mathit{pref}(i,j) = \mathit{pref}(s_i, s_j)$ be the string associated with edge (i,j).

As examples we draw in Figure 1 the overlap graph and the distance graph for our previous example $S_0 = \{$ ate, half, lethal, alpha, alfalfa $\}$. All edges not shown have overlap 0. Note that the sum of the distance and overlap weights on an edge (i,j) is the length of the string s_i.

Notice now that $\mathrm{TSP}(G_S) \leq \mathrm{OPT}(S) - ov(\mathit{last}(s), \mathit{first}(s)) \leq \mathrm{OPT}(S)$, where $\mathrm{TSP}(G_S)$ 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 $\mathit{last}(s)$ for only $d(\mathit{last}(s), \mathit{first}(s))$ instead of its full length.

We now define some notation for dealing with directed cycles in G_S. 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 $t = vu$. If c is a directed cycle in G_S with vertices $i_0, \ldots, i_{r-1}$ in order around c, we define $\mathit{strings}(c)$ to be the equivalence class $[\mathit{pref}(i_0, i_1)\mathit{pref}(i_1, i_2) \cdots \mathit{pref}(i_{r-1}, i_0)]$ and $\mathit{strings}(c, i_k)$ the rotation starting with $\mathit{pref}(i_k, i_{k+1})$, *i.e.*, the string $\mathit{pref}(i_k, i_{k+1}) \cdots \mathit{pref}(i_{k-1}, i_k)$, where subscript arithmetic is modulo r. Let us say that an equivalence class $[s]$ has *periodicity* k ($k > 0$), if s is invariant under a rotation by k characters ($s = uv = vu$, $|u| = k$). Obviously, $[s]$ has periodicity $|s|$. A moment's reflection shows that the minimum periodicity of $[s]$ must equal the number of distinct rotations of s. This is the size of the equivalence class and denoted by $\mathit{card}([s])$. Furthermore, it is easily proven that if $[s]$ has periodicities a and b, then it has periodicity $\gcd(a,b)$ as well. (See, *e.g.*, [4].) It follows that all periodicities are a multiple of the minimum one. In particular, we have that $|s|$ is a multiple of $\mathit{card}([s])$.

In general, we will denote a cycle c with vertices $i_1, \ldots, i_r$ in the order by "$i_1 \rightarrow \cdots \rightarrow i_r \rightarrow i_1$." Also, let $w(c)$, the *weight* of cycle c, equal $|s|$, $s \in strings(c)$. 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 $c = i_0 \rightarrow \cdots \rightarrow i_{r-1} \rightarrow i_0$ and c' be cycles in G_S. For any string s, s^k denotes the string consisting of k copies of s concatenated together.

Claim 2 *Each string s_{i_j} in c is a substring of s^k for all $s \in strings(c)$ 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). Taking $k = \lceil |s_{i_j}|/w(c) \rceil$ and $l = kr$ we get that s_{i_j} is a prefix of $pref(i_j, i_{j+1}) \cdots pref(i_{j+kr-1}, i_{j+kr}) = strings(c, i_j)^k$, which itself is a substring of s^{k+1} for any $s \in strings(c)$. ■

Claim 3 *If each of $\{s_{j_1}, \ldots, s_{j_r}\}$ is a substring of s^k for some string $s \in strings(c)$ and sufficiently large k, then there exists a cycle of weight $|s| = w(c)$ containing all these 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 $\{s_{j_1}, \ldots, s_{j_r}\}$ and the strings in c whose successive distances sum to $|s|$. ■

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

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

Claim 5 *If $strings(c') = strings(c)$, 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 $card(strings(c))$ containing all vertices in c.*

Proof. Let u be the prefix of length $card(strings(c))$ of some string $s \in strings(c)$. By our periodicity arguments, $|u|$ divides $|s| = w(c)$, and $s = u^j$ where $j = w(c)/|u|$. It follows that every string in $strings(c) = [s]$ is a substring of u^{j+1}. Now use Claim 3 for u. ■

The following lemma has been proved in [15, 16]. Figure 2 below gives a graphical interpretation of it. In the figure, the vertical bars surround pieces of string that match, showing a possible overlap between v^- and u^+, giving an upper bound on $d(v^-, u^+)$.

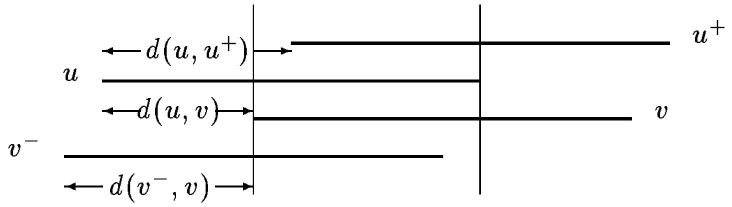

Figure 2: Strings and overlaps

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. The conditions in the above Lemma are also known as "Monge conditions" in the context of transportation problems [1, 3, 7]. In this sense the Lemma follows from the observation that optimal shipping routes do not intersect. In the string context, we are transporting 'items' from the ends of substrings to the fronts of substrings.

3 A $4 \cdot$ 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* [10]. 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 these particular graphs, a greedy strategy can be used to find optimal assignments. This result can also be found (in a somewhat different form) as Theorem 1 in Hoffman's 1963 paper [7].

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 $c_i = i_1 \to \cdots \to i_r \to i_1$, 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 different cycles of C. The lemma is also implied by the results in [4].

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 $x = strings(c)$ and $x' = strings(c')$. Since C is a minimum weight assignment, we know $x \neq x'$. Otherwise, by Claim 5, we could find a lighter assignment by combining the cycles c and c'. In addition, by Claim 6, $w(c) \leq card(x)$.

Suppose that s and s' overlap in a string u with $|u| \geq w(c) + w(c')$. Denote the substring of u starting at the i-th symbol and ending at the j-th as $u_{i,j}$. Since by Claim 2 $s = t^k$ for some $t \in x$ and large enough k and $s' = t'^{k'}$ for some $t' \in x'$ and large enough k', we have that $x = [u_{1,w(c)}]$ and $x' = [u_{1,w(c')}]$. ¿From $x \neq x'$ we conclude that $w(c) \neq w(c')$; assume without loss of generality that $w(c) > w(c')$. Then

$$u_{1,w(c)} = u_{1+w(c'),w(c)+w(c')} = u_{1+w(c'),w(c)} u_{w(c)+1,w(c)+w(c')} = u_{1+w(c'),w(c)} u_{1,w(c')}.$$

This shows that x has periodicity $w(c') < w(c) \leq card(x)$, which contradicts the fact that $card(x)$ is the minimum periodicity. ■

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,

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

We are now ready to present the algorithm MGREEDY, and show that it in fact 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 in the decreasing order of their overlaps (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 the head (or tail, respectively) of f. By the definition of MGREEDY, it includes an edge f if and only if it has not yet included an edge dominating f.

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 $e = k \to j$ is in $M \setminus N$. The two N edges $i \to j$ and $k \to l$ that share head and tail with e are not in M, and thus are dominated by e. Since $ov(k,j) \geq \max\{ov(i,j), ov(k,l)\}$, by Lemma 7, $ov(i,j) + ov(k,l) \leq ov(k,j) + ov(i,l)$. Thus replacing in N these two edges with $e = k \to j$ and $i \to l$ would yield an assignment N' that has more edges in common with M and has no less overlap than N. This would contradict our choice of N. ■

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 $3 \cdot \mathbf{OPT}(S)$

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 $S = \{s_1, \ldots, s_m\}$ 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. For any set R of strings, define $||R|| = \sum_{x \in R} |x|$ to be the total length of the strings in set R. Also let $w = \sum_{x \in T} w_x$. Since $\mathrm{CYC}(G_S) \leq \mathrm{TSP}(G_S) \leq \mathrm{OPT}(S)$, we have $w \leq n$.

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 [15, 16], 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, $|s| \leq n_T + w$.

For each $x \in T$, let s_{i_x} be the string in cycle c_x that is a prefix of x. Let $S' = \{s_{i_x} | x \in T\}$, $n' = \mathrm{OPT}(S')$, $S'' = \{strings(c_x, i_x)s_{i_x} | x \in T\}$, 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, we have $|S''_\pi| \leq |S'_\pi| + \sum_{x \in T} w_x$, so $n'' \leq n' + w$, where S'_π and S''_π are the superstrings obtained by overlapping the members of S' and S'', respectively, in the order given by π. Observe that $S' \subseteq S$ implies $n' \leq n$. Summing up, we get

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

Combined with $|s| \leq n_T + w$, 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* $4 \cdot OPT(S)$.

Before proving the theorem formally, we give a sketch of the basic idea behind the proof. If we want to relate the merges done by GREEDY to an optimal assignment, we have to keep track of what happens when GREEDY violates the maximum overlap principle, i.e. when some self-overlap is better than the overlap in GREEDY's merge. One thing to try is to charge GREEDY some extra cost that reflects that an optimal assignment on the new set of strings (with GREEDY's merge) may be somewhat longer than the optimal assignment on the former set (in which the self-overlapping string would form a cycle). If we could just bound these extra costs then we would have a bound for GREEDY. Unfortunately, this approach fails because the self-overlapping string may be merged by GREEDY into a larger string which itself becomes self-overlapping, and this nesting could go arbitrarily deep. Our proof concentrates on the inner-most self-overlapping strings only. These so called culprits form a linear order in the final superstring. We avoid the complications of higher level self-overlaps by splitting the analysis in two parts. In one part, we ignore all the original substrings that connect first to the right of a culprit. In the other part, we ignore all the original substrings that connect first to the left of a culprit. In each case, it becomes possible to bound the extra cost. This method yields a bound of $7 \cdot \mathrm{OPT}(S)$. By combining the two analyses in a more clever way, we can even eliminate the effect of the extra costs and obtain the same $4 \cdot \mathrm{OPT}(S)$ bound as we found for MGREEDY. A detailed formal proof follows.

Proof of Theorem 12. We will need some notions and lemmas. 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. (w.l.o.g., the strings are renumbered to reflect their order in the superstring produced by GREEDY.) For convenience we will usually abbreviate s_i to i. GREEDY does not include an edge f iff

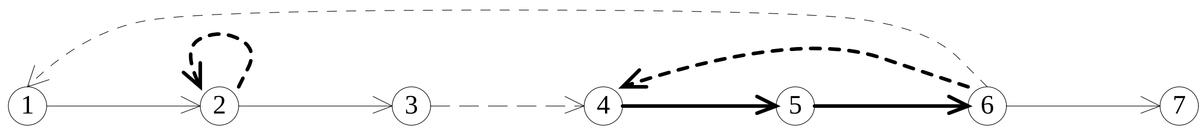

Figure 3: Culprits and weak links in Greedy merge path.

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 $f = j \to i$ necessarily has $i \leq j$. Each bad back edge $f = j \to i$ 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 $i-1 \to i$ and $j \to j+1$. The bad back edge f is said to *span* the closed interval $I_f = [i, j]$. The above observations provide a proof of 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 and bad back edges do not cross each other. *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, because of the minimality of the culprits, if $f = j \to i$ 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. The example in Figure 3 has 7 substrings, of which 2 by itself and the merge of 4, 5, and 6 form the culprits (indicated by thicker lines). Bad back edges are $2 \to 2$, $6 \to 4$, and $6 \to 1$. The weak link $3 \to 4$ is the worst edge between culprits [2] and [4, 5, 6]. The blocks in this example are thus [1, 2, 3] and [4, 5, 6, 7], and we have $S_l = \{1\}, S_m = \{2, 4, 5, 6\}, S_r = \{3, 7\}$.

The following lemma shows that a bad back edge must be from a middle or right node to a middle or left node.

Lemma 14 *Let $f = j \to i$ 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 $i-1 \to i$. 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 then implies 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. (Here x^R denotes the reversal of string x.) 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 $cost(E_r) = l_r + l_c$.

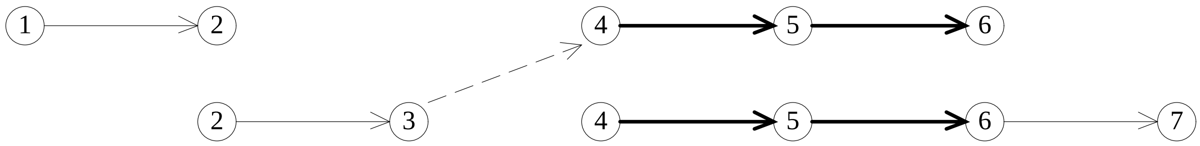

Figure 4: Left/middle and middle/right parts with weak links.

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. Our basic idea is to charge the overlap of each edge of M to an edge of E or a weak link or the back edge of a culprit in a way such that every edge of E and every weak link is charged at most once and the back edge of each culprit is charged at most twice. This is achieved through combining the left/middle and middle/right parts carefully as shown below. For convenience, we will refer to the union operation for multisets (*i.e.*, allowing duplicates) as the *disjoint union.*

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 $ov(E') = ov(E_l) + ov(E_r) + o_w$. A picture of (V, E') for our previous example is given in Figure 4.

Let M be the disjoint union of M_l and M_r, an assignment on graph G. 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 $e = t \to h$ 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. (Note that if e is in fact the back edge of some culprit, then both cases 2 and 3 hold.)

Suppose that e does not correspond to a bad back edge. Then e must be dominated by some greedy edge since it was not chosen by GREEDY. If the greedy edge dominating 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. Since N already has the first two properties, it suffices to argue that if N violates property 3, then we can construct another assignment N' that satisfies properties 1 and 2, and has more edges in common with E'.

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

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 w_c the summed weight of all culprit cycles, i.e., the weight of the (optimal) assignment C_m on S_m from Lemma 15. Then $l_c = w_c + o_c$. As in the proof of Theorem 8, we have $o_c - 2w_c \leq n$ and $w_c \leq n$. (Note that the overlap of a culprit back edge is less than the length of the longest string in the culprit cycle.) Putting everything together, the string produced by GREEDY has length

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

■

6 Which algorithm is the best?

Having proved various bounds for the algorithms GREEDY, MGREEDY, and TGREEDY, one may wonder what this implies about their relative performance. First of all we note that MGREEDY can never do better than TGREEDY since the latter applies the GREEDY algorithm to an intermediate set of strings that the former merely concatenates.

Does the $3n$ bound for TGREEDY then mean that it is the best of the three? This proves not always to be the case. In the example $\{c(ab)^k, (ab)^{k+1}a, (ba)^k c\}$, GREEDY produces the shortest superstring $c(ab)^{k+1}ac$ of length $n = 2k+5$, whereas TGREEDY first separates the middle string to end up with something like $c(ab)^k ac(ab)^{k+1}a$ of length $4k+6$.

Perhaps then GREEDY is always better than TGREEDY, despite the fact that we cannot prove as good an upper bound for it. This turns out not to be the case either, as shown by the following example. On input $\{cab^k, ab^kab^ka, b^kdab^{k-1}\}$, TGREEDY separates the middle string, merges the other two, and next combines these to produce the shortest superstring $cab^kdab^kab^ka$ of length $3k+6$, whereas GREEDY merges the first two, leaving nothing better than $cab^kab^kab^kdab^{k-1}$ of length $4k+5$.

Another greedy type of algorithm that may come to mind is one that arbitrarily picks any of the strings and then repeatedly merges on the right the string with maximum overlap. This algorithm, call it NAIVE, turns out to be disastrous on examples like

$$\{abcde, bcde\#a, cde\#a\#b, de\#a\#b\#c, e\#a\#b\#c\#d, \#a\#b\#c\#d\#e\}.$$

Instead of producing the optimal $abcde\#a\#b\#c\#d\#e$, NAIVE might pick $\#a\#b\#c\#d\#e$ as a starting point to produce $\#a\#b\#c\#d\#e\#a\#b\#c\#de\#a\#b\#cde\#a\#bcde\#abcde$. It is clear that in this way superstrings may be produced whose length grows quadratically in the optimum length n.

7 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 problems in bounded-degree graphs, max cut, etc. This is considered rather unlikely.[1]

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 $|cost(x) - opt(a)| \leq \beta |cost(y) - opt(b)|$.

[1] In fact, Arora et al. [2] have recently shown that MAX SNP-hard problems do not have polynomial time approximation schemes, unless P = NP.

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 [11]. 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 [12] 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. Without loss of generality, assume that each vertex of H has outdegree at least 2. The reduction is similar to the one of [5] 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 $2m+3n+k+1$. 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 $\sum_{i=1}^{n}(2 + 2d_i + 1) + k + 1 = 3n + 2m + k + 1$.

Conversely, suppose we are given a common superstring of length $3n + 2m + k + 1$. 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 [15, 16]. That is, maximizing the total compression is also MAX SNP-hard.

8 Open problems

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

(1) Obtain an algorithm which achieves a performance better than 3 times the optimum.

(2) Prove or disprove the conjecture that GREEDY achieves 2 times the optimum.

9 Acknowledgments

We thank Samir Khuller and Vijay Vazirani for discussions on the superstring algorithms (Samir brought the authors together), and Rafi Hassin for bringing Hoffman's and other's work on Monge sequences to our attention. We would also like to thank the referees for their helpful comments.

References

[1] N. Alon, S. Cosares, D. Hochbaum, and R. Shamir. An algorithm for the detection and construction of Monge Sequences. *Linear Algebra and its Applications 114/115, 669–680, 1989.*

[2] A. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and hardness of approximation problems. *Proc. 33rd IEEE Symp. Found. Comp. Sci.*, 1992, 14-23.

[3] E. Barnes and A. Hoffman. On transportation problems with upper bounds on leading rectangles. *SIAM Journal on Algebraic and Discrete Methods 6, 487–496, 1985.*

[4] N. Fine and H. Wilf. Uniqueness theorems for periodic functions. *Proc. Amer. Math. Soc.* 16, 1965, 109-114.

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

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

[7] A. Hoffman. On simple transportation problems. *Convexity: Proceedings of symposia in pure mathematics, Vol 7, American Mathematical Society, 317–327, 1963.*

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

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

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

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

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

[13] 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.*

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

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

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

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