The Greedy Algorithm for Shortest Superstrings

Haim Kaplan * Nira Shafrir *

August 26, 2004

Keywords: algorithms, analysis of algorithms, approximation algorithms, shortest superstring

1 Introduction

In the shortest superstring problem the input is a set $S = s_1, \cdots, s_n$ of strings and we seek the shortest possible string s such that every string in S is a (contiguous) substring of s. This problem is known to be NP-hard and even MAX-SNP hard [3]. The best known approximation algorithm finds a string whose length is at most 2.5 times the length of the optimal string [6, 8]. The superstring problem has applications in data compression and in DNA sequencing. For example, in a shotgun DNA sequencing, a DNA molecule can be represented as a string over the set of nucleotides $\{A, C, G, T\}$. Only Small overlapping fragments of the DNA molecule can be sequenced at a time. So the DNA sequence has to be reconstructed from these fragments. We can model this as a shortest superstring problem in which each string in S represents a sequenced DNA fragment, and a shortest superstring of S is the DNA sequence representation for the whole DNA molecule.

There is a natural greedy algorithm for the shortest superstring problem, which we refer to as GREEDY. The GREEDY algorithm maintains a set of strings, initialized to be equal to S. At each iteration GREEDY picks two strings with maximum overlap from its set, combines them into one string, which it then puts back into the set. Blum et. al. [3] proved that the length of the string produced by GREEDY is within a factor of 4 from optimal. Blum et. al. [3] also gave the following example that shows that GREEDY produces a sequence twice as long as the optimal. $S = \{c(ab)^k, (ba)^k, (ab)^k c\}$. It's easy to see that GREEDY will first join $c(ab)^k$ and $(ab)^k c$ to create the string $c(ab)^k c$, and then it'll concatenate $c(ab)^k c$ with $(ba)^k$ to get the string $c(ab)^k c(ba)^k$ of length $4k+2$. The optimal superstring however, is $c(ab)^{k+1}c$ and its length is $2k+4$. Blum et. al. [3] conjectured that indeed the approximation guarantee of GREEDY is 2.

*School of Computer Science, Tel-Aviv University, Tel Aviv, Israel. E-mail: {shafrirn,haimk}@post.tau.ac.il.
This research was supported in part by the Israel Science Foundation (ISF) (grant no. 548).

String Length	Number of Blocks	Average Length of the superstring produced by GREEDY.
250	50	381
400	80	596
450	90	677
500	100	768

Table 1: The first column is the random string length, the second column is the number of blocks (number of input strings for GREEDY), the last column is the average length of the superstring produced by GREEDY. In the first two rows the average is computed over 50 random strings, in the last two rows the average is over 20 random strings.

Zaritsky and Sipper [10] have recently presented results of experiments that support the conjecture that the string produced by GREEDY is of length at most 2 times the length of the shortest superstring. Zaritsky and Sipper compared the performance of several algorithms for the shortest superstring problem to that of the GREEDY algorithm. In their experiments they generated a random binary string, took 5 disjoint copies of it, and divided each copy into blocks (substrings) of random length between 20 and 30 bits. The blocks were the input to the superstring algorithms. Since the string was randomly generated it is likely that the length of the shortest superstring of these blocks is close to the length of the original random string. The results of these experiments with respect to the GREEDY algorithm are presented in Table 1. In all of their experiments the average length of the string produced by GREEDY is less than 2 times the length of the random string.

Despite a considerable progress achieved since the work of Blum et. al. on designing approximation algorithms other than GREEDY with better approximation guarantees than [4], (see [9, 5, 7, 1, 2, 4, 8, 6]), there has been no progress since on narrowing the gap regarding the approximation guarantee of GREEDY. Many of the approximation algorithms for the superstring problem are more complicated and less efficient than GREEDY. This makes it important to know whether the approximation ratio of GREEDY is in fact better than that of the other approximation algorithms.

In this paper we prove that the string produced by GREEDY is in fact within a factor of 3.5 from the optimal string. To get this result, we use the "overlap-rotation lemma" of Breslauer, Jiang and Jiang (see [4], Section 3), to tighten the upper bound on the total length of the so called "culprit bad back edges" in the proof of Blum et. al. [3].

2 Preliminaries

We assume that no string in S contains another and denote the length of the shortest superstring by $OPT(S)$. We first give some basic definitions. For two strings s and t, we call $ov(s,t)$ the amount of overlap of s with respect to t. I.e. $ov(s,t)$ is the length of the longest string y such that $s = xy$ and $t = yz$ for non-empty strings x and z. We define $pref(s,t)$ to be the string x, and $d(s,t)$ to be $|pref(s,t)|$. These definitions imply that $|s| = d(s,t) + ov(s,t)$.

We denote by $s =< s_{i_1}, \cdots, s_{i_k} >$ the string $pref(s_{i_1}, s_{i_2})pref(s_{i_2}, s_{i_3}) \cdots pref(s_{i_{k-1}}, s_{i_k})s_{i_k}$. String s is a superstring of $s_{i_1}, \cdots, s_{i_k}$. In fact s is the shortest superstring string in which $s_{i_1}, \cdots, s_{i_k}$ appear such that s_{i_j} starts before $s_{i_{j+1}}$ for every $1 \leq j \leq k-1$. So clearly, the shortest superstring of S must be $< s_{i_1}, \cdots, s_{i_n} >$, for some permutation of the strings $s_1, \cdots, s_n$. Notice that, $OPT(S) = \sum_{j=1}^{n} |s_{i_j}| - \sum_{j=1}^{n-1} ov(s_{i_j}, s_{i_{j+1}})$, and we get the following equality.

$$\sum_{i=1}^{n} |s_i| = OPT(S) + \sum_{j=1}^{n-1} ov(s_{i_j}, s_{i_{j+1}}). \tag{1}$$

With this terminology the definition of GREEDY is as follows. We initialize a set of strings to contain all the input strings. In each iteration we remove s and t $(s \neq t)$ from S where s and t are such that $ov(s,t)$ is as large as possible and add back to S the string $< s,t >= pref(s,t)t$. Greedy terminates when there is only one string in S.

The distance graph $G_S = (V, E, w)$ is a complete directed graph $(E = \{(u,v) | u, v \in V\})$. The set of vertices V is the set of strings, $s_1, \cdots, s_n$. The weight of an edge (s_i, s_j), is $d(s_i, s_j)$. The *overlap graph* is similar to the distance graph where the weight of an edge (s_i, s_j) is $ov(s_i, s_j)$ rather than $d(s_i, s_j)$. Let $c = s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ be a cycle in G_S. Cycle c corresponds to the string $pref(s_{i_1}, s_{i_2})pref(s_{i_2}, s_{i_3}) \cdots pref(s_{i_{k-1}}, s_{i_k})pref(s_{i_k}, s_{i_1})$. The weight of c in G_S is $w(c) = d(s_{i_1}, s_{i_2}) + d(s_{i_2}, s_{i_3}) + \cdots + d(s_{i_k}, s_{i_1})$. Notice that,

$$| < s_{i_1}, \cdots, s_{i_k} > | = d(s_{i_1}, s_{i_2}) + d(s_{i_2}, s_{i_3}) + \cdots + d(s_{i_{k-1}}, s_{i_k}) + |s_{i_k}|$$

$$= d(s_{i_1}, s_{i_2}) + d(s_{i_2}, s_{i_3}) + \cdots + d(s_{i_{k-1}}, s_{i_k}) + d(s_{i_k}, s_{i_1}) + ov(s_{i_k}, s_{i_1}) = w(c) + ov(s_{i_k}, s_{i_1}).$$

Let $s =< s_{i_1}, \cdots, s_{i_n} >$ be the shortest superstring of S, and let $c' = s_{i_1}, s_{i_2}, \ldots, s_{i_n}$ be the corresponding cycle in G_S, then $w(c') = OPT(S) - ov(s_{i_n}, s_{i_1})$. It follows that if $TSP(G_S)$ is the cost of a minimum weight Hamiltonian cycle in G_S, then $TSP(G_S) \leq w(c') \leq OPT(S)$. We denote the minimum weight cycle cover (CC) in a graph G by $CYC(G)$. Then we have that $w(CYC(G_S)) \leq TSP(G_S) \leq OPT(S)$.

We now define the *factor* of a string and other related terms that are used in [4]. For a string s, $factor(s)$ is the shortest string x such that $s = x^i y$, and y is a prefix of x, (y may be empty).

We denote $period(s)$ to be $|factor(s)|$. A semi-infinite string $s = a_1a_2\cdots$ is *periodic* if $s = xs$ for some nonempty string x. The shortest string x such that $s = xs$ is the *factor* of s, denoted by $factor(s)$. As for finite strings we also define $period(s) = |factor(s)|$. The semi-infinite strings we use in the followings lemmas are obtained by breaking a cycle c at a certain position j. Let $c = s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ be a cycle in G_S, then we if we break the cycle at some index j we get a string $u = pref(s_{i_j}, s_{i_{j+1}})pref(s_{i_{j+1}}, s_{i_{j+2}})\cdots pref(s_{i_{k-1}}, s_{i_k})pref(s_{i_k}, s_{i_1})\cdots pref(s_{i_{j-1}}, s_{i_j})$. The string $uuuu\cdots$ is the semi-infinite string obtained from the cycle c by breaking it at position j.

Let s and t be two strings either finite or periodic semi-infinite, we call s and t *equivalent* if $factor(t)$ is a cyclic shift $factor(s)$. I.e. there are strings x, y such that $factor(s) = xy$ and $factor(t) = yx$. Otherwise they are *inequivalent*. The following lemmas are proved in [3], and were restated in [4].

Lemma 2.1 *Let* $c = s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ *be a cycle in* $CYC(G_S)$*, then* $factor(< s_{i_1}, \cdots, s_{i_k} >) = factor(< s_{i_1}, \cdots, s_{i_k}, s_{i_1} >) = pref(s_{i_1}, s_{i_2})\cdots pref(s_{i_{k-1}}, s_{i_k})pref(s_{i_k}, s_{i_1})$*, and* $period(< s_{i_1}, \cdots, s_{i_k}, s_{i_1} >) = w(c)$.

Notice that the strings s_{i_j} *on the cycles* c *are not necessarily equivalent, to each other and to* $< s_{i_1}, \cdots, s_{i_k} >$.

Lemma 2.2 *Let* $c = s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ *and* $c' = s_{j_1}, s_{j_2}, \ldots, s_{j_l}$ *be two different cycles in* $CYC(G_S)$*, then* $< s_{i_1}, \cdots, s_{i_k} >$ *is inequivalent to* $< s_{j_1}, \cdots, s_{j_l} >$.

The following lemma was used to get the bounds in [3].

Lemma 2.3 *If* s, t *are inequivalent then* $ov(s, t) \leq period(s) + period(t)$.

Lemma 2.1, Lemma 2.2 and Lemma 2.3 imply that if $c = s_{i_1}, s_{i_2}, \ldots, s_{i_k}$ and $c' = s_{j_1}, s_{j_2}, \ldots, s_{j_l}$ are two different cycles in $CYC(G_S)$, then

$$ov(< s_{i_1}, \cdots, s_{i_k} >, < s_{j_1}, \cdots, s_{j_l} >) \leq w(c) + w(c'). \tag{2}$$

Given a periodic semi-infinite string $\alpha = a_1a_2, \ldots$, let $\alpha[k]$ be the string $a_ka_{k+1}\cdots$. We now state the overlap rotation lemma proved in [4].

Lemma 2.4 *[overlap rotation lemma] Let* α *be a periodic semi-infinite string. There exists an integer* $k \leq period(\alpha)$ *such that for any finite string* s *that is inequivalent to* α

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

The following Lemma is also from [4]. It shows based on the overlap rotation lemma how we can derive a string from each cycle in the cycle cover such that these strings have pairwise small overlaps (smaller than the overlap in Inequality (2)).

Lemma 2.5 *Let $c = s_{i_1}, \cdots, s_{i_r}$ be a cycle in $CYC(G_S)$, then there is a string t_c and an index j such that*

1. *The string $< s_{i_{j+1}}, \cdots, s_{i_r}, s_{i_1}, \cdots, s_{i_j} >$ is a suffix of the string t_c.*
2. *String t_c is contained in $s_c =< s_{i_j}, \cdots, s_{i_r}, s_{i_1}, \cdots, s_{i_j} >$*
3. *$factor(t_c)$ is equivalent to $factor(< s_{i_{j+1}}, \cdots, s_{i_r}, s_{i_1}, \cdots, s_{i_j} >)$.*
4. *The semi-infinite string $u = factor(t_c)factor(t_c)\cdots$ is the desired string $\alpha[k]$ in Lemma 2.4, where α is any semi-infinite string defined by the cycle c. Specifically, Let x be the string produced by this lemma that corresponds to a different cycle $c' \in CYC(G_S)$, then $ov(x, t_c) \leq w(c') + \frac{1}{2}w(c)$.*

The next lemma gives a bound on the length of the superstring for the set that consists of strings chosen according to Lemma 2.5 for each cycle in $CYC(G_S)$. It easily follows from Property 2 in Lemma 2.5.

Lemma 2.6 *Let $A = \{t_c \mid c \in CYC(G_s)\ t_c$ is the string guaranteed to exist by to Lemma 2.5$\}$. Then $OPT(A) \leq OPT(S) + w(CYC(G_s))$.*

Proof: Let $B = \{s_c \mid c \in CYC(G_s)\ s_c$ is chosen according to Property 2 of lemma 2.5$\}$. Since each string $t_c \in A$ is contained in the string $s_c \in B$, a superstring for the set B is also a superstring of the set A. In [3], Blum et. al. proved that $OPT(B) \leq OPT(S) + w(CYC(G_s))$, so it follows that $OPT(A) \leq OPT(B) \leq OPT(S) + w(CYC(G_s))$. □

We now mention some definitions and details from the 4-approximation proof of GREEDY in [3]. For this proof, Blum et. al. [3] introduced another algorithm called MGREEDY and proved that it finds a minimum cycle cover on G_s. MGREEDY is similar to GREEDY except that when there is a string $t \in S$ such that $ov(t,t) > ov(s',t')$ for any two strings $s', t' \in S$, MGREEEDY removes t from S, and continues with the rest of the strings. Each of the strings extracted from S in this process defines a cycle in a minimum cycle cover.

Blum et. al. looked at the GREEDY algorithm as taking a list of all edges in the overlap graph sorted in decreasing order by their overlap, and going down the list deciding for each edge whether to include it or not. For the rest of the paper assume that the strings are renumbered such that if s'

is the string produced by GREEDY, then $s' =< s_1, s_2, \cdots, s_n >$. We say that an edge e *dominates* an edge f if e comes before f in the sorted list (I.e. $ov(e) \geq ov(f)$), and e share its head or tail with f. It is easy to verify that GREEDY doesn't include an edge f if either

1. f is dominated by an already chosen edge e.
2. f is not dominated but would form a cycle with the strings already chosen.

If f wasn't chosen because of the latter reason, then f is a *bad back edge*. Let $f = (s_j, s_i)$ be a bad back edge, then since f closes a cycle with the strings already chosen by GREEDY, f corresponds to a string $< s_i, s_{i+1}, \cdots, s_j >$. When GREEDY considers f it had already chosen all edges on the path from i to j. Thus the overlap of all edges on the path from i to j is greater or equal to $ov(f)$. Also, when GREEDY considers f it had not yet chosen the edges (s_{i-1}, s_i) and (s_j, s_{j+1}). (Otherwise f wouldn't be chosen because of the first reason). We say that the edge f *spans* the closed interval $I_f[i, j]$. Blum et. al. proved the following lemma.

Lemma 2.7 *Let e and f be two bad back edges. Then the closed intervals I_e, and I_f, spanned by e and f respectively are either disjoint or one contains the other.*

Thus intervals of bad back edges do not cross each other. *Culprits* are minimal (with respect to containment) such intervals. By definition, the intervals of all culprits are disjoint. Each culprit $[i, j]$ corresponds to a string $< s_i, \cdots, s_j >$. As mentioned above the edge (s_j, s_i) has the lowest overlap among the edges of the culprit $(s_i, s_{i+1}), \cdots, (s_{j-1}, s_j)$. Let $S_m \subseteq S$ be the set of all strings that belong to culprits. I.e. for each culprit $[i, j]$, $s_i, s_{i+1}, \cdots, s_j \in S_m$. Let C_m be the cycle cover over S_m defined by the culprits. I.e. each culprit $[i, j]$, defines the cycle $c_{ij} = s_i, s_{i+1}, \cdots, s_j$ and $w(c_{ij}) = d(s_i, s_{i+1}) + d(s_{i+1}, s_{i+2}) + \cdots + d(s_{j-1}, s_j) + d(s_j, s_i)$. It is straightforward to see that if we apply MGREEDY to the subgraph of the distance graph induced by S_m the algorithm will construct the cycle cover C_m. This implies that C_m is a minimum cycle cover in the distance graph defined by the set S_m. We denote by w_c the weight of the cycle cover C_m. Since $S_m \subseteq S$, and C_m is a minimum cycle cover over S_m, we have that $w_c \leq OPT(S_m) \leq OPT(S)$. We denote by o_c the sum of the overlaps of all culprits back edges. I.e. $o_c = \sum_{[i,j]=culprit} ov(s_j, s_i)$.

3 A bound of 3.5 on the approximation ratio

Blum et. al. [3] showed that the length of the string t produced by GREEDY is

$$|t| \leq 2OPT(S) + o_c - w_c. \tag{3}$$

To get the 4-approximation they used Inequality (2) to bound o_c. Using Inequality (2), they showed that $o_c \leq OPT(S) + 2w_c$, which together with Inequality (3) implies an upper bound of 4 on the approximation ratio. We get our result by proving that $o_c \leq OPT(S) + 1.5w_c$, which together with Inequality (3) implies a bound of 3.5 on the approximation ratio.

Let $c_{ir} = s_1, \cdots s_r$ be the cycle in C_m that corresponds to the culprit $[i, r]$. Let $a_{ir} = t_{c_{ir}}$ be the string guaranteed to exist for that cycle by Lemma 2.5. Let $A = \{a_{ir} \mid [i, r] \text{ is a culprit } \}$. The following is a key lemma in establishing our result.

Lemma 3.1 *For each culprit* $[i, r]$, $ov(s_r, s_i) \leq |a_{ir}| - w(c_{ir})$

Proof: Let $j \in [i, r]$ be such that by Lemma 2.5, a_{ir} contains $< s_{j+1}, \cdots, s_r, s_1, \cdots, s_j >$ as a suffix. Since the length of the overlap between s_r and s_i is the smallest among all the overlaps of strings in c_{ir}, we have that $ov(s_r, s_i) \leq ov(s_j, s_{j+1})$. String a_{ir} contains $< s_{j+1}, \cdots, s_j >$, so,

$$\begin{aligned}
|a_{ir}| &\geq |< s_{j+1}, \cdots, s_j >| \\
&= |pref(s_{j+1}, s_{j+2}) \cdots pref(s_{j-1}, s_j) s_j| \\
&= |pref(s_{j+1}, s_{j+2}) \cdots pref(s_{j-1}, s_j) pref(s_j, s_{j+1})| + ov(s_j, s_{j+1}) \\
&\geq w(c_{ir}) + ov(s_r, s_i).
\end{aligned}$$

□

Finally we establish the bound we claimed on o_c.

Lemma 3.2 *We have that* o_c *the total overlap of the bad back edges is at most* $OPT(S) + 1.5w_c$.

Proof: We denote by $|A|$ the sum of the lengths of all the strings in A, then $|A| = \sum_{[i,r]=culprit} |a_{ir}|$. We denote by $OV(A)$ the total overlap between adjacent strings in the shortest superstring of A. I.e. if $< b_1, \cdots, b_k >$ is the shortest superstring of A, then $OV(A) = \sum_{j=1}^{k-1} ov(b_j, b_{j+1})$. Let $c_i \in C_m$ be the cycle that corresponds to the string $b_i \in A$. Using Lemma 2.5,we get that

$$OV(A) = \sum_{j=1}^{k-1} ov(b_j, b_{j+1}) \leq \sum_{j=1}^{k-1} w(c_j) + \frac{1}{2} w(c_{j+1}) \leq 1.5w_c \tag{4}$$

Recall that by Equality (1), $|A| = OPT(A) + OV(A)$. By Lemma 2.6, $OPT(A) \leq OPT(S_m) + w_c \leq OPT(S) + w_c$. Putting it all together we get that,

$$|A| = OPT(A) + OV(A) \leq OPT(S) + w_c + 1.5w_c = OPT(S) + 2.5w_c.$$

By Lemma 3.1 for each culprit $[i, r]$, $ov(s_r, s_i) \leq |a_{ir}| - w(c_{ir})$. Summing over all culprits we get that $o_c \leq |A| - w_c$, and we get that $o_c \leq |A| - w_c \leq OPT(S) + 1.5w_c$. □

References

[1] C. Armen, C. Stein. Improved length bounds for the shortest superstring problem. Proc. of the *4th International Workshop on Algorithms and Data Structures (WADS)*, 494-505, 1995.

[2] C. Armen, C. Stein. A 2 2/3-approximation algorithm for the shortest superstring problem. Proc. of the *7th Annual Symposium on Combinatorial Pattern Matching (CPM)*, 87-101, 1996.

[3] A. Blum, T. Jiang, M. Li, J. Tromp and M. Yannakakis, Linear Approximation of Shortest Superstring. *J. of the ACM*, 31(4), 1994, 630-647.

[4] D. Breslauer, T. Jiang and Z. Jiang, Rotations of periodic strings and short superstrings, *Journal of Algorithms* 24(2):340–353, 1997.

[5] A. Czumaj, L. Gasieniec, M. Piotrw, W. Rytter. Sequential and parallel approximation of shortest superstrings. *Journal of Algorithms* 23(1):74-100, 1997.

[6] H. Kaplan, M. Lewenstwein, N. Shafrir, and M. Sviridenko. Approximation algorithms for asymmetric tsp by decomposing directed regular multigraphs. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, page 56 , 2003.

[7] S.R. Kosaraju, J.K. Park and C. Stein. Long tours and short superstrings. *Proc. 35th Annual Symposium on Foundations of Computer Science (FOCS)*, 166-177, 1994.

[8] Z. Sweedyk, $2\frac{1}{2}$-approximation algorithm for shortest superstring, *SIAM J. Comput.* 29(3):954-986, 2000.

[9] S.H. Teng and F. Yao. Approximating shortest superstrings. *SIAM Journal on Computing*, 26(2), 410-417, 1997.

[10] A. Zaritsky, M. Sipper The Preservation of Favoured Building Blocks in the Struggle for Fitness: The Puzzle Algorithm . To appear in *IEEE Transactions on Evolutionary Computation.*