A GREEDY APPROXIMATION ALGORITHM FOR CONSTRUCTING SHORTEST COMMON SUPERSTRINGS*

Jorma Tarhio and Esko Ukkonen
Department of Computer Science, University of Helsinki
Tukholmankatu 2, SF-00250 Helsinki, Finland

Abstract. An approximation algorithm for the shortest common superstring problem is developed, based on the Knuth-Morris-Pratt string matching procedure and on the greedy heuristics for finding longest Hamiltonian paths in weighted graphs. Given a set R of strings, the algorithm constructs a common superstring for R in **O**(mn) steps where m is the number of strings in R and n is the total length of these strings. The performance of the algorithm is analyzed in terms of the compression in the common superstrings constructed, that is, in terms of n – k where k is the length of the obtained superstring. We show that $(n - k) \geq (n - k_{min}) / 2$ where k_{min} is the length of a shortest common superstring. Hence the compression achieved by the algorithm is at least half of the maximum compression. It also seems that the lengths always satisfy $k \leq 2 \cdot k_{min}$ but proving this remains open.

1. Introduction

The shortest common superstring problem is, given a finite set R of strings over some (finite or infinite) alphabet Σ, to find a shortest string w such that each string x in R is a substring of w, that is, w can be written as uxv for some u and v. Since the decision version of this problem is NP-complete [3, 6, 2], we are interested in polynomial time approximation algorithms. Such algorithms construct a superstring w which is not necessarily a shortest one.

An obvious approach is to develop a "greedy" approximation algorithm based on the following idea [2]: Find and remove two strings in R which have the longest mutual overlap amongst all possible pairs in R. Then form the overlapped string from the removed two strings and replace it back in R. Repeat this until there is only one string in R or no two strings have a nonempty overlap.

In this paper we develop an algorithm that implements this idea. The algorithm uses, besides the greedy heuristics, also a modification of the Knuth-Morris-Pratt string matching procedure to find the pairwise overlaps between the strings in R. It runs in **O**(mn) steps where m is the number of strings in R and n is the total length of these strings.

Our main result deals with the performance of the algorithm, i.e., the quality of approximation. We show that the "compression" in the superstring constructed by the algorithm is at least half of the compression in a shortest superstring. That is, if n is the total length of the original strings in R and k_{min} is the length of a shortest superstring and k is the length of the superstring computed by the approximation algorithm, then $(n - k) \geq (n - k_{min}) / 2$. Another natural way to measure the quality is to directly compare lengths k and k_{min}. In all examples we have been able to construct we have $k \leq 2 \cdot k_{min}$. Proving that this is always true remains open.

* The work was supported by the Academy of Finland.

It turns out that the shortest common superstring problem can be understood as a special case of the longest Hamiltonian path problem for weighted directed graphs. The results of Jenkyns [4] imply that for arbitrary weighted graphs the greedy heuristics finds a Hamiltonian path whose length is at least one third of the length of a longest path. Here we have special graphs encoding the possible pairwise maximal overlaps of the strings in R. For such graphs, the greedy heuristics finds a Hamiltonian path whose length is at least half of the length of a longest path.

Superstring algorithms can be used in compressing data. Molecular biology is another application area. The primary structure of a biopolymer such as a DNA-molecule can be represented as a character string which typically is a few thousands of symbols long. To determine this string for different molecules, or to sequence the molecules, is a crucial step towards understanding the biological functions of the molecule. Unfortunately, with the current laboratory methods it is impossible to sequence a long molecule as a whole. Rather, the sequencing proceeds piecewise by determining relatively short fragments (typically less than 500 characters long) of the long string. From the fragments whose location in the long string is more or less arbitrary and unknown, one has to assemble the long string representing the whole molecule. Without computer assistance the assembly task becomes intolerably tedious since there may be hundreds of fragments that altogether contain thousands of symbols.

The assembly problem can be modeled as a shortest common superstring problem: The fragments are the strings in set R and the superstring constructed represents an approximation to the original long string. Although there is no a priori reason to aim at a shortest superstring this seems the most natural restriction that makes the problem nontrivial. Moreover, our experience shows that programs based on shortest common superstring algorithms work very satisfactorily for real biological data [7].

2. The approximation algorithm

The following simple property of shortest common superstrings is useful in deriving the approximation method.

Lemma 1. If a string x is a substring of another string y in R (i.e. y = uxv for some strings u and v) then sets R and $R - \{x\}$ have the same shortest common superstrings.

A set of strings R is called reduced if no string in R is a substring of another string in R. Assume in what follows that R is reduced. Later on it will be shown that it is easy to form a reduced R from an arbitrary R. By Lemma 1, this reduction preserves the shortest common superstrings.

There is an overlap v between strings x and x', $x \neq x'$, if v is both a suffix of x and a prefix of x', that is, x = uv and x' = vu'. Note that v can be the empty string and that if x and x' are from a reduced set, neither of strings u and u' can be empty. Also note the antisymmetry of our definition: An overlap v between x and x' is not necessarily an overlap between x' and x. An overlap v will be called maximal if it is longest possible.

Let $R = \{x_1, ..., x_m\}$ be reduced. Define the overlap graph for R as follows. The graph is a directed graph with node set $R \cup \{x_0, x_{m+1}\}$, and with directed arcs (x_i, x_j), $i \neq j$. Here the augmenting node x_0 is called the start node and the augmenting node x_{m+1} the end node. Each arc (x_i, x_j) has a weight w_{ij} where w_{ij} is the length of the maximal overlap between strings x_i and x_j. In particular, $w_{0j} = 0$ for $j > 0$ and $w_{j\,m+1} = 0$ for $j < m+1$

Consider any directed Hamiltonian path H from x_0 to x_{m+1} in the overlap graph for R. From H we may construct a common superstring for R: Just place strings in R above each other in the order of appearance on path H. The successive x_i's must be maximally overlapped, that is, the length of the overlap equals the weight of corresponding arc on the path. If $y_1, ..., y_m$ are the elements of R written in the order they appear on path H from x_0 to x_{m+1}, we get an arrangement as shown in Figure 1. Clearly, the "projection" p(H) of strings y_i gives a common superstring for R.

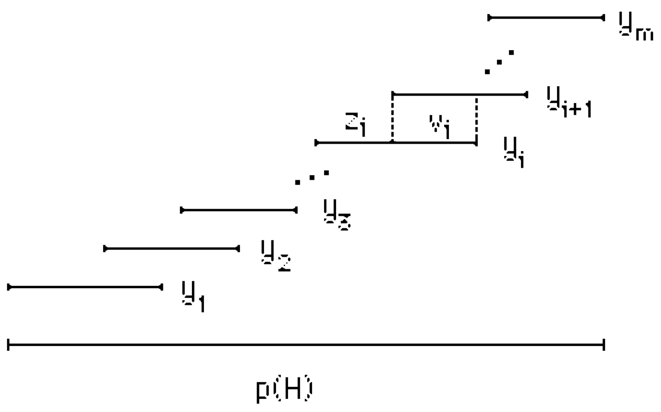


Figure 1.

More carefully, let v_i be the maximal overlap between y_i and y_{i+1}. Then y_i can be written as $y_i = z_i v_i$ and p(H), the common superstring for R constructed from H, is $z_1 z_2 ... z_{m-1} y_m$.

The length of p(H) is $(|x_1| + ...+ |x_m|) - |H|$ where |H| describes the sum of the weights on path H. Thus the the larger is |H|, the shorter is the corresponding common superstring. This suggests that a shortest common superstring for R would be a shortest p(H), that is, p(H) where H is longest possible.

To prove that a shortest common superstring q cannot be shorter than a shortest p(H), it suffices to show that q has a decomposition similar to that of p(H) shown in Figure 1 and that the pairwise overlaps in that decomposition are maximal. Therefore, let $x_1, x_2, ..., x_m$ be the strings in R indexed in the left-to-right order in which their occurrence in q starts. Let u_i be the unique substring of q that is covered both by x_i and x_{i+1}. Hence there is an overlap u_i between x_i and x_{i+1}. Moreover, let t_i be the substring of q that is covered by x_i but not by x_{i+1}. So the situation is like in Figure 1, but with p(H) replaced by q and with y_i's, v_i's and z_i's replaced by x_i's, u_i's and t_i's, respectively. Now we have:

Lemma 2. The maximal overlap between each x_i and x_{i+1} is u_i.

Proof. Assume that for some i, the maximal overlap between x_i and x_{i+1} is u and $|u| > |u_i|$. Let $x_i = t_i u$. Then $t_1...t_{i-1}t'_i t_{i+1}...t_{m-1}x_m$ is a common superstring of R which is shorter than $q = t_1...t_{i-1}t_i t_{i+1}...t_{m-1}x_m$, a contradiction.

We have shown the following connection between shortest common superstrings and longest Hamiltonian paths:

Theorem 3. A shortest common superstring for R is string p(H) where H is a longest Hamiltonian path from the start node to the end node in the overlap graph for R.

The NP-hardness of finding a shortest common superstring implies that also finding the Hamiltonian path in Theorem 3 is NP-hard. (The NP-hardness of finding maximal Hamiltonian paths in arbitrary weighted graphs is well-known, of course [3]; here we only have the additional remark that the NP-hardness is preserved if the problem is restricted to the class of overlap graphs.) Hence we have to look at approximation algorithms. We will use the following well-known "greedy" heuristics for longest Hamiltonian paths:

To construct a Hamiltonian path, select an arc e from the remaining arcs of the overlap graph such that

(i) e has the largest weight; and

(ii) e together with the arcs selected earlier forms a subgraph which can be expanded to a Hamiltonian path from the start node to the end node;

and repeat this until a Hamiltonian path from the start node to the end node has been constructed.

The above discussion suggests an algorithm for computing an approximate shortest common superstring whose main steps can be summarized as follows:

A1. Compute the maximal pairwise overlaps between all strings in R;

A2. Form reduced R by removing all strings x which are substrings of some other string. It turns out that this can be accomplished together with step A1. By Lemma 1, reducing R does not change the set of the shortest common superstrings of R;

A3. Using the greedy heuristics, find an approximation to the longest Hamiltonian path from the start node to the end node in the overlap graph for reduced R. From the path H found in this way, construct a common superstring p(H) for R.

In the subsequent three subsections we refine the details of the steps A1 – A3 as well as analyze their time requirements. The quality of the approximation is analyzed in Section 3.

2.1. Step A1: Finding maximal pairwise overlaps

The problem to be solved here is, given two strings x and x', to determine the length k of the maximal overlap between x and x'. If x' is a substring of x then k = |x'|. Else k = |v| where v is the longest string such that x = uv and x' = vu' for some u and u'.

This problem can be understood as a generalization of the classical pattern matching problem where x is the text and x' is the pattern and we ask whether the pattern occurs in the text. If the pattern does not occur in the text we now also ask what is the longest suffix of the text which is also a prefix of the pattern.

The classical problem can be solved with Knuth-Morris-Pratt algorithm (KMP-algorithm) in time $\mathbf{O}(|x'| + |x|)$, i.e., linear in the length of the strings involved [5]. Not surprisingly, the KMP-algorithm is immediately seen to solve, still in linear time, also our generalized problem:

Recall that the KMP-algorithm works in two phases. In the preprocessing phase the algorithm constructs from the pattern x' a so-called pattern matching machine. Preprocessing takes time $\mathbf{O}(|x'|)$. The pattern matching machine is, in fact, a finite-state automaton. The automaton has $|x'| + 1$ states, numbered 0, 1, ..., $|x'|$. In the second phase, the automaton scans the text x, spending time $\mathbf{O}(|x|)$. The state transitions proceed during scanning such that the automaton is in state i whenever i is the largest index such that $a_1...a_i$ is both a suffix of the scanned portion of x and a prefix of $x' = a_1a_2...a_{|x'|}$. Therefore the automaton enters state $|x'|$ if and only if x contains an occurrence of x'. It also follows that after scanning the whole x the automaton is in state i if and only if $a_1...a_i$ is the longest suffix of x which is also a prefix of x'. Hence i is the length of the maximal overlap between x and x', which means that the KMP-algorithm can be used in implementing step A1.

This way the maximal overlaps between any two strings x_i and x_j in R can be found in time $\mathbf{O}(|x_i| + |x_j|)$. By constructing the pattern matching machine for each x_i and by scanning with the machine all the remainig strings, all pairwise overlaps can be found in total time $\mathbf{O}(mn)$.

2.2. Step A2: Forming reduced R

This step is naturally implemented by embedding into step A1: Whenever step A1 finds out that x_i is a substring of x_j, remove x_i from R, and proceed to the next x_i. This does not increase the asymptotic time requirement of step A1.

2.3. Step A3: Finding an approximation to the longest Hamiltonian path

The greedy heuristics for finding an approximate longest Hamiltonian path runs by repeatedly finding an arc with maximal weight which together with the arcs selected earlier can be expanded to a Hamiltonian path from the start node to the end node. Hence it is forbidden to select an arc that is not any more free. An arc (x, y) is called free if x is not the start node of some arc selected earlier and y is not the end node of some arc selected earlier and (x, y) together with the arcs selected earlier does not create an oriented cycle.

In more detail, let H be the set of arcs selected to the Hamiltonian path. Initially H is empty. We proceed as follows:

1. Sort the arcs according to the weight.
2. Scan the arcs in decreasing order. For each arc (x, y) encountered, if (x, y) is free then
   - 2.1. add (x, y) to H, and
   - 2.2. mark x to an start node and y to an end node (this makes all remaining arcs starting from x or ending at y nonfree), and
   - 2.3. in order to find the ends of the path in H that contains the new arc (x, y), traverse in H the path from y until the end node y' is encountered and traverse the path from x against the direction of the arcs until the start node x' is encountered and then mark the arc (y', x') to an cycle-creating arc.

Step 1 can be efficiently implemented by bucket sort in time $\mathbf{O}(m^2)$. Using bucket sort is reasonable since the weights to be sorted are in the limited range 0... n. Space requirement is $\mathbf{O}(n + m^2)$.

Step 2 requires time **O**(m^2) for scanning the arcs and for testing their freeness. The markings made in steps 2.2 and 2.3 ensure that the freeness of each arc can be tested in constant time. Hence the total time for steps 2.1 and 2.2 is **O**(m). Each application of step 2.3 takes time **O**(|H|), hence total time **O**(m^2).

The total time for step A3 is therefore **O**(m^2) = **O**(mn).

Noting finally that p(H) is easily constructed from H in time **O**(n), we can summarize our analysis:

Theorem 4. The algorithm in steps A1 – A3 can be implemented such that its time requirement is **O**(mn) where m is the number of strings in R and n is the total length of the strings.

3. A performance guarantee

Approximation algorithms for NP-complete optimization problems find solutions which are feasible but not necessarily optimal. The approximation algorithm of Section 2 constructs a string p(H) which is a superstring for R but not always shortest possible. The length of p(H) equals n – |H| where n is the total length of strings in R and |H| is the length of the underlying Hamiltonian path constructed by our algorithm. Hence |H| is the "compression" in p(H) as compared to the trivial approximate solution obtained by catenating the original strings.

Path H was constructed by the greedy heuristics for finding maximal Hamiltonian paths. Finding such paths is closely related to finding maximal travelling salesman tours. From a result in [4] it follows that the greedy method satisfies $|H| \geq |H_{max}| / 3$ where H_{max} is the longest possible path. For overlap graphs we get the stronger result in Theorem 6. First we obtain an important technical property of overlap graphs.

If two arcs r and s of the overlap graph have a common start node (i.e. a start string), then we write $r \leftarrow\!\!\rightarrow s$. In the same way, $r \rightarrow\!\!\leftarrow s$ denotes that r and s have a common end node.

The weight of an arc r is denoted by w(r).

A string x is said to have self-overlap v, if x = vu = u'v for some nonempty strings v, u, u'.

Lemma 5. (a) If $t \leftarrow\!\!\rightarrow r$, $t \rightarrow\!\!\leftarrow s$ such that $w(r) \leq w(t)$ and $w(s) \leq w(t)$ and the end node of r is not the start node of s, then there is an arc q such that $q \leftarrow\!\!\rightarrow s$, $q \rightarrow\!\!\leftarrow r$ and $w(q) \geq w(r) + w(s) - w(t)$.

(b) Let t, r, s be as in (a), but assume that the end node z of r is the start node of s. Then string z has a self-overlap of length $\geq w(r) + w(s) - w(t)$.

Proof. (a) The lemma is trivially true when $w(r) + w(s) \leq w(t)$. Hence assume $w(r) + w(s) > w(t)$. Let string $a_1 \ldots a_{w(t)}$ correspond to the arc t. Then $a_{w(t)-w(r)+1} \ldots a_{w(t)}$ corresponds to r and $a_1 \ldots a_{w(s)}$ corresponds to s. But then there is an overlap $a_{w(t)-w(r)+1} \ldots a_{w(s)}$ between the start string of s and the end string of r, that is, the corresponding arc q has weight $\geq w(r) + w(s) - w(t)$. The situation is illustrated in Figure 2, where t = (x, y), r = (x, z), s = (z', y).

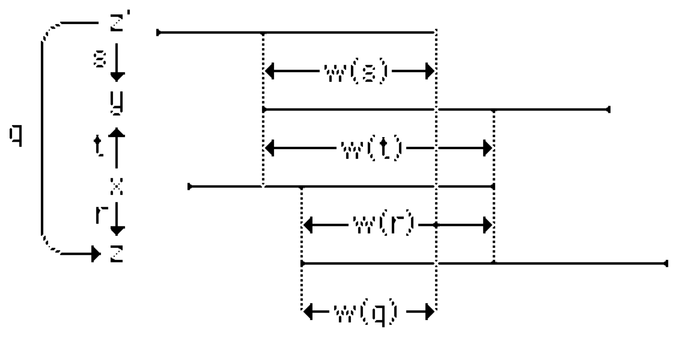


Figure 2.

(b) The situation is as in (a) but now z = z'. Hence z has a self-overlap $a_{w(t)-w(r)+1} \ldots a_{w(s)}$.

Theorem 6. Let H be the approximate longest Hamiltonian path constructed by the greedy heuristics for the overlap graph of R, and let H_{max} be the longest Hamiltonian path. Then $|H| \geq |H_{max}| / 2$.

Proof. Let H consist of arcs $t_1, t_2, \ldots, t_{m+1}$, listed in the order in which the greedy algorithm selects them. Consider the algorithm at the moment it has selected ith arc, t_i, to H and made the appropriate arcs nonfree. Let $H_i = \{t_1, \ldots, t_i\}$. Initially H_0 is empty. Moreover, let K_i denote a set of arcs containing all arcs in H_{max} that are still free and some other free arcs whose freeness we will be able to infer using Lemma 5, as follows:

Initially $K_0 = H_{max}$.

In general,

$$K_i = (K_{i-1} \setminus \{r, s, p, t_i\}) \cup \{q\}. \qquad (1)$$

Here r is the arc in K_{i-1} such that $t_i \leftarrow r$. If K_{i-1} does not contain such an arc, then r is missing. Similarily, s is the arc in K_{i-1} such that $t_i \rightarrow s$. If K_{i-1} does not contain such an arc, then s is missing. Arc p is the arc in K_{i-1} which together with the arcs in H_i forms an oriented cycle. Again, if K_{i-1} does not contain such an arc, then p is missing. Arcs r, s, p are all arcs in K_{i-1} which are made nonfree by the selection of t_i. Hence they are removed from K_{i-1} to get K_i. Of course, t_i has to be removed since it is added to H_{i-1} to get H_i. Arc q is inferred by Lemma 5, as will be explained in a moment.

At this stage it is easy to understand the lower bound of Jenkyns [4]. Each greedy selection eliminates at most three correct arcs and the selected arc has at least as large weight as the eliminated ones. For example, if H_{max} contains arcs (x_1, x_2), (x_2, x_3) and (x_3, x_4) and the algorithm selects arc (x_3, x_2), then all three correct arcs are lost. This means, a path with at least one third of the maximal total weight can be found.

To obtain a better lower bound for the overlap graphs we apply Lemma 5. Whenever r and s exist in (1) and the end node z of r differs from the start node z' of s then by Lemma 5 (a), there must be an arc q = (z', z) such that

$$w(t_i) + w(q) \geq w(r) + w(s) \qquad (2)$$

Moreover, if q is still free (it is not free only if it forms a cycle with the arcs in H_i), it is added to K_i in (1). Otherwise q is missing in (1).

It turns out, that arcs q compensate one of the eliminated arcs r, s, p such that on the average, the weight of only two arcs is lost. Unfortunately, the proof will be rather complicated. The difficulty is that whenever q is nonfree a greedy step really can eliminate three arcs. However, this is possible only if the loss in some earlier step was essentially smaller.

By induction it is easy to show that no two arcs in $H_i \cup K_i$ start at the same node or end at the same node. Let G_i denote the subgraph with arcs $H_i \cup K_i$. Then the following is true for i = 0, ..., m + 1:

G1. Subgraph G_i consists of disjoint directed paths and cycles.

A cycle cannot contain arcs only from H_i since H_i must be extendible to a Hamiltonian path. Also a cycle with only one arc p from K_i is impossible since such a p cannot be free. Therefore:

G2. Each cycle in G_i contains at least two arcs from K_i.

Let denote by c_i the number of cycles in G_i . Now we claim that

$$2 \cdot |H_i| + |K_i| - c_i \cdot w(t_i) \geq |H_{max}|. \qquad (3)$$

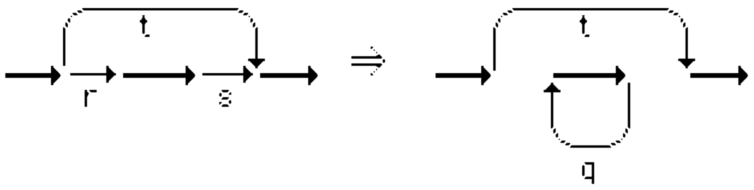
The proof is by induction on i.

Let first i = 0. Since $K_0 = H_{max}$, H_0 is empty and $c_0 = 0$, claim (3) is true.

Assume then that (3) holds for i – 1 and consider i, i > 0. Depending on how $t = t_i = (x, y)$ is selected we have the following cases.

Case 1. Nodes x and y belong to the same directed path of G_{i-1}, x is before y, arc q exists in (1).

The situation is depicted in the following figure, where the diagram before the arrow represents graph $G_{i-1} \cup \{t\}$ and after the arrow graph G_i. Bold edges denote directed (possible empty) paths, thin edges single arcs.



Obviously, $c_i = c_{i-1} + 1$ and $K_i = (K_{i-1} \setminus \{r, s\}) \cup \{q\}$. Recalling (2) and noting that $w(t) \leq w(t_{i-1})$, we can estimate the left hand side of (3):

$$\begin{aligned} &2 \cdot |H_i| + |K_i| - c_i \cdot w(t) \\ &\quad = 2 \cdot |H_{i-1}| + 2 \cdot w(t) + |K_{i-1}| + w(q) - w(r) - w(s) - c_{i-1} \cdot w(t) - w(t) \\ &\quad \geq 2 \cdot |H_{i-1}| + |K_{i-1}| - c_{i-1} \cdot w(t) \\ &\quad \geq 2 \cdot |H_{i-1}| + |K_{i-1}| - c_{i-1} \cdot w(t_{i-1}) \end{aligned}$$

which is $\geq |H_{max}|$ by induction hypotesis.

Case 2. Same as Case 1 but q missing in (1).

Arc q can be missing for two different reasons. First, there is no r or s different from t. Then actually t = r = s, and t must be in K_{i-1}. Since now $c_i = c_{i-1}$, we obtain:

$$\begin{aligned} &2 \cdot |H_i| + |K_i| - c_i \cdot w(t) \\ &\quad = 2 \cdot |H_{i-1}| + 2 \cdot w(t) + |K_{i-1}| - w(t) - c_{i-1} \cdot w(t) \\ &\quad \geq 2 \cdot |H_{i-1}| + |K_{i-1}| - c_{i-1} \cdot w(t_{i-1}) \\ &\quad \geq |H_{max}|, \end{aligned}$$

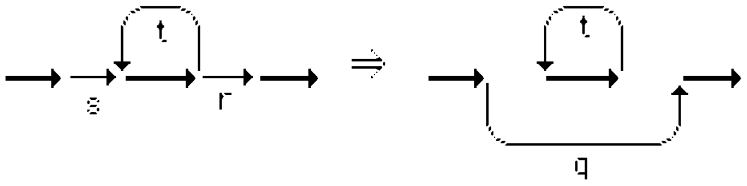
as required.

The second possibility is that although r and s exist and are different, arc q is not free any more. Again $c_i = c_{i-1}$. Since $w(t) \geq w(r)$ and $w(t) \geq w(s)$, we get:

$$\begin{aligned}
&2 \cdot |H_i| + |K_i| - c_i \cdot w(t) \\
&\qquad = 2 \cdot |H_{i-1}| + 2 \cdot w(t) + |K_{i-1}| - w(r) - w(s) - c_{i-1} \cdot w(t) \\
&\qquad \geq 2 \cdot |H_{i-1}| + |K_{i-1}| - c_{i-1} \cdot w(t_{i-1}) \\
&\qquad \geq |H_{max}|.
\end{aligned}$$

Case 3. Nodes x and y belong to the same directed path of G_{i-1}, y is before x, t creates a cycle.

The situation is as follows:



Now $c_i = c_{i-1} + 1$. It can be shown, exactly as in Case 1, that (3) is true.

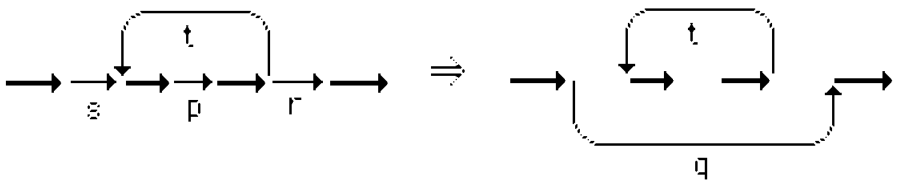
It is also possible that s and the path leading to s or r and the path after r is missing, in which case q is missing, too. For example, if s is missing, we get

$$\begin{aligned}
&2 \cdot |H_i| + |K_i| - c_i \cdot w(t) \\
&\qquad = 2 \cdot |H_{i-1}| + 2 \cdot w(t) + |K_{i-1}| - w(r) - c_{i-1} \cdot w(t) - w(t) \\
&\qquad \geq 2 \cdot |H_{i-1}| + |K_{i-1}| - c_{i-1} \cdot w(t_{i-1}) \\
&\qquad \geq |H_{max}|.
\end{aligned}$$

The other cases are similar.

Case 4. Same as Case 3 but t does not create a cycle.

The situation is as follows:



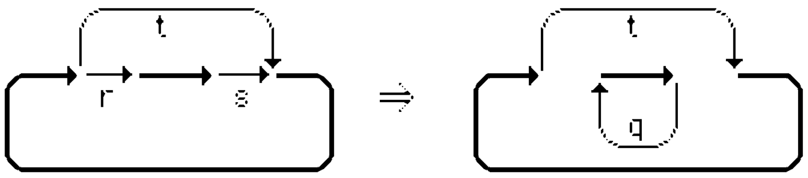
Then $c_i = c_{i-1}$ and $K_i = (K_{i-1} \setminus \{r, s, p\}) \cup \{q\}$. Using (2) we obtain

$$\begin{aligned}
&2 \cdot |H_i| + |K_i| - c_i \cdot w(t) \\
&\qquad = 2 \cdot |H_{i-1}| + 2 \cdot w(t) + |K_{i-1}| + w(q) - w(r) - w(s) - w(p) - c_{i-1} \cdot w(t) \\
&\qquad \geq 2 \cdot |H_{i-1}| + |K_{i-1}| - c_{i-1} \cdot w(t) \\
&\qquad \geq |H_{max}|.
\end{aligned}$$

Again s or r can be missing in which case also q is missing. The above estimates are easily adapted to these cases.

Case 5. Nodes x and y are on the same cycle of G_{i-1}. Arc t belongs to a cycle in G_i.

If q exists in (1), the situation is as follows:

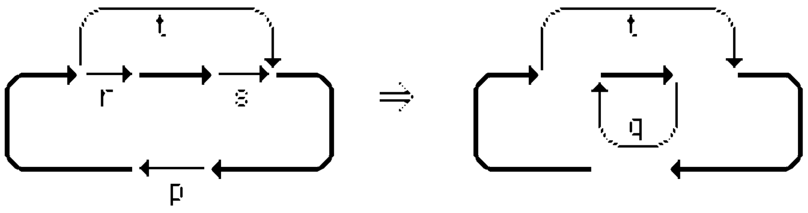


Then $c_i = c_{i-1} + 1$ and $K_i = (K_{i-1} \setminus \{r, s\}) \cup \{q\}$ and (3) is shown exactly as in Case 1.

If q does not exist, the proof proceeds as in Case 2.

Case 6. Same as Case 5, but t does not belong to a cycle in G_i.

Then the original cycle in G_{i-1} disappears since arc p becomes non-free. Pictorially, when q exists:



Obviously $c_i = c_{i-1}$ and $K_i = (K_{i-1} \setminus \{r, s, p\}) \cup \{q\}$, and (3) is shown exactly as in Case 4.

When q is missing, a new type of situation arises since then $c_i = c_{i-1} - 1$. If q is missing because t = r = s (and hence t is in K_{i-1}), we get:

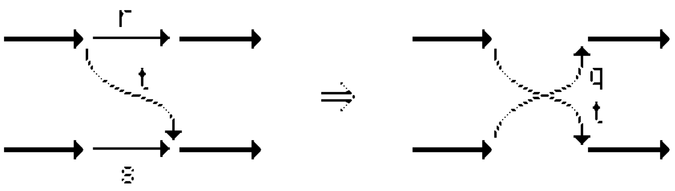
$$\begin{aligned} &2 \cdot |H_i| + |K_i| - c_i \cdot w(t) \\ &\quad = 2 \cdot |H_{i-1}| + 2 \cdot w(t) + |K_{i-1}| - w(t) - w(p) - c_{i-1} \cdot w(t) + w(t) \\ &\quad \geq 2 \cdot |H_{i-1}| + |K_{i-1}| - c_{i-1} \cdot w(t_{i-1}) \\ &\quad \geq |H_{max}|. \end{aligned}$$

If q is missing because it is not free although r and s exist and are different, then we obtain:

$$\begin{aligned} &2 \cdot |H_i| + |K_i| - c_i \cdot w(t) \\ &\quad = 2 \cdot |H_{i-1}| + 2 \cdot w(t) + |K_{i-1}| - w(r) - w(s) - w(p) - c_{i-1} \cdot w(t) + w(t) \\ &\quad \geq 2 \cdot |H_{i-1}| + |K_{i-1}| - c_{i-1} \cdot w(t_{i-1}) \\ &\quad \geq |H_{max}|. \end{aligned}$$

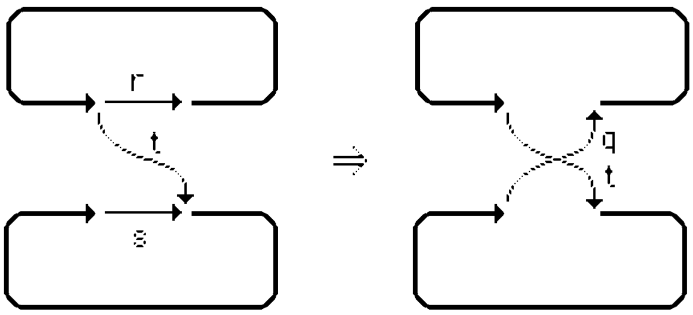
Case 7. Nodes x and y are in different components of G_{i-1}.

Assume first that x and y belong to different paths. Then the situation is as follows:



Obviously, $c_i = c_{i-1}$ and $K_i = (K_{i-1} \setminus \{r, s\}) \cup \{q\}$. The proof of (3) is as in Case 4 but now p is missing. The proof is easily further adapted to the cases where r or s and hence q are missing.

If x and y belong to different cycles, the situation is as follows:



Since both original cycles contain at least two arcs from K_{i-1} (property G2), q is always free and therefore $K_i = (K_{i-1} \setminus \{r, s\}) \cup \{q\}$. Now $c_i = c_{i-1} - 1$. The proof of (3) is as in the last part of Case 6 but without arc p. The proof is easily adapted to the cases where either of the original cycles is a path.

Since Cases 1 – 7 cover all possible situations, we have proven the inductive step.

To complete the proof of the theorem, consider the final stage where the algorithm has built a Hamiltonian path $H_{m+1} = H$ from x_0 to x_{m+1}. Properties G1 and G2 imply that K_{m+1} must be empty and $c_{m+1} = 0$. Then (3) gives $2 \cdot |H| \geq |H_{max}|$, as required.

The next example shows that the result of Theorem 6 is best possible.

Example 7. Let $R = \{ab^h, b^hb, b^ha\}$. Then the shortest common superstring is ab^hba while the greedy heuristics may produce superstring ab^hab^hb. Maximal compression $|H_{max}|$ is 2h while the greedy method may give $|H| = h$. Note that also the greedy method can yield the shortest common superstring depending on which one of the arcs with equal weights is selected first.

Another natural question on the performance is how much longer can the approximate superstring be than the shortest one? The worst example we have found is as in Example 7 where the shortest string is of length h + 3 and the approximate string is of length 2h + 3, which is less than twice the shortest length. This also shows that the claim in [2] that the length k of the approximate string is always at most 3/2 times the shortest length k_{min} cannot be true.

Suggested by our example we conjecture that always $k \leq 2 \cdot k_{min}$. Theorem 6 implies that the conjecture is true for all sets R for which $|H_{max}| \leq 2n / 3$. For other sets the conjecture remains open.

On the other hand, let us make the relatively strong restsriction on R that the overlap graph contains no cycles consisting only of arcs with weights > 0 and that no string in R has a non-empty self-overlap. For such R the greedy algorithm works accurately and produces a shortest common superstring. The proof is similar to the proof of Theorem 6. Instead of (3), one should prove inequality

$$|H_i| + |K_i| \geq |H_{max}|.$$

4. Experiments and modifications

We have carried out some experiments with the greedy algorithm. On the average, the superstring obtained was only one percent longer than a shortest superstring. In the worst observed case, the length increase was about 15 percent. The length of our random test strings, which were mainly in the binary alphabet, varied from 4 to 100. The algorithm worked, of course, better with longer strings, because the maximal overlaps among them are relatively shorter.

A problem with the greedy algorithm is that it may do selections that forbid later use of good overlaps. Therefore we have experimented with some additional heuristics, which try to eliminate such selections. The following one seems the most promising. The basis is the same as in the greedy algorithm, but the weights of arcs are computed differently. The weights are updated after every selection, and if there are several arcs having the same maximal new weight, the selection among them is based on the original weights. After each selection the new weight W(t) for each arc t = (x, y) is computed as

$$\begin{aligned} W(t) = k \cdot w(x, y) & \\ & - \max \{w(x, y') \mid (x, y') \text{ is free, } y \neq y'\} \\ & - \max \{w(x', y) \mid (x', y) \text{ is free, } x' \neq x\} \end{aligned}$$

Here w(u,v) denotes, as before, the length of the overlap between u and v.

The idea is to take in the consideration also the arcs which would become non-free if t is selected next. Parameter k can be used to tune the method. In our experiments the best results were obtained with parameter values from 2 to 2.5. Then the error of the modified algorithm was on the average about 1/5 of that of the original algorithm.

Building shortest common superstrings can be understood as a data compression problem. This suggests that our algorithm can also be modified to compress sparse matrices [1, 8], such as parser tables. In such matrices only a few entries store significant values. Assume that R consists of the rows (or the columns) of such a matrix. Then the rows can be overlapped such that identical significant entries overlap or a significant and an insignificant entry overlap. Unfortunately, the Knuth-Morris-Pratt algorithm cannot be used to compute such overlaps. A slower method has to be used. Moreover, the overlap structure is not static: using an overlap may make adjacent maximal overlaps shorter. The necessary modifications to our algorithm should be quite obvious after which the method produces a superstring for R, that is, a compressed matrix.

5. Concluding remarks

We have shown that the greedy algorithm for shortest common superstrings has a performance ratio ≥ 1/2 when the amount of compression is the performance measure.

There is a dual view on the problem. Instead of maximal Hamiltonian paths in the overlap graph, one could base the algorithm on finding minimal Hamiltonian paths in "non-overlap" graphs. In such a graph, arc (x, y) has weight

$$|x| - w(x, y).$$

The dual heuristics is to successively select the smallest of the remaining free arcs until a Hamiltonian path is constructed. The following example (due to P. Orponen) shows, however, that this method can behave essentially worse that the original algorithm. Let R contain strings

$$\begin{array}{ll} a_0a_1 & a_1t^n \\ a_1a_2 & a_2a_1t^{n-1} \\ \vdots & \vdots \\ a_{n-1}a_n & a_na_{n-1}\ldots a_1t \end{array}$$

Then the shortest common superstring is

$$a_0a_1...a_{n-1}a_na_{n-1}...a_1t^n$$

The length is 3n. This string is also formed by the original approximation method. The dual method can produce string

$$a_0a_1t^na_1a_2a_1t^{n-1}...a_{n-1}a_na_{n-1}a_1t$$

with length n (n + 2) and compression n. The perfomance ratio is (n + 2) / 3 for the length and 1 / n for the compression .

Acknowledgement. The authors are indebted to one of the referees for the suggestion to use bucket sort.

References

1. P. Dencker, K. Dürre and J. Heuft: Optimization of parser tables for portable compilers. ACM TOPLAS **6** (Oct. 1984), 546–572.
2. J. K. Gallant: String compression algorithms. Ph.D. Thesis, Princeton University, 1982.
3. M. R. Garey and D. S. Johnson: Computers and Intractability. W. H. Freeman, 1979.
4. T. A. Jenkyns: The greedy travelling salesman's problem. Networks **9** (1979), 363–373.
5. D. Knuth, J. Morris and V. Pratt: Fast pattern matching in strings. SIAM J. Comput. **6** (1977), 323–350.
6. D. Maier and J. A. Storer: A note on complexity of the superstring problem. TR–233, Princeton University, Dept. EECS, 1977.
7. H. Peltola, H. Söderlund, J. Tarhio and E. Ukkonen: Algorithms for some string matching problems arising in molecular genetics. Information Processing 83 (Proceedings of the IFIP Congress 83), pp. 53–64. North-Holland, 1983.
8. R. E. Tarjan and A. C. Yao: Storing a sparse table. Comm. ACM **22** (Nov. 1979), 606–611.