Combined Super-/Substring and Super-/Subsequence Problems

Martin Middendorf[1] and David F. Manlove[2]

[1] *Parallel Computing and Complex Systems Group, University of Leipzig, D-04109 Leipzig, Germany. Email:* middendorf@informatik.uni-leipzig.de.

[2] *Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK. Email:* davidm@dcs.gla.ac.uk.

Abstract

Super-/substring problems and super-/subsequence problems are well known problems in stringology that have applications in a variety of areas, such as manufacturing systems design and molecular biology. Here we investigate the complexity of a new type of such problem that forms a combination of a super-/substring and a super-/subsequence problem. Moreover we introduce different types of minimal superstring and maximal substring problems. In particular, we consider the following problems: given a set L of strings and a string S, (i) find a minimal superstring (or maximal substring) of L that is also a supersequence (or a subsequence) of S, (ii) find a minimal supersequence (or maximal subsequence) of L that is also a superstring (or a substring) of S. In addition some non-super-/non-substring and non-super-/non-subsequence variants are studied. We obtain several NP-hardness or even MAX SNP-hardness results and also identify types of "weak minimal" superstrings and "weak maximal" substrings for which (i) is polynomial-time solvable.

1 Introduction

Super-/substring and super-/subsequence problems for sets of strings find important applications in many areas, including project and process planning, manufacturing systems design and computational molecular biology (see e.g. [18]). This is because many objects in nature can be modeled as strings and the super-/substring and super-/subsequence relations are the most basic and natural relations between strings. Several papers study super-/substring and super-/subsequence problems with respect to their complexity (see e.g. [3, 5, 9, 10, 14, 17, 18]). Four well-studied problems are the Shortest Common Superstring (SCSt), Longest Common Substring (LCSt), Shortest Common Supersequence (SCSe), and Longest Common Subsequence (LCSe) problems. Each of SCSt, SCSe and LCSe is known to be MAX SNP-hard [1] (see [16] for the theory of MAX SNP-hardness). SCSt is polynomial-time solvable for strings of length 2 and becomes NP-hard for strings of length 3 [5]. SCSt is also NP-hard for an alphabet of size 2 [5]. LCSt is solvable in polynomial time (see e.g. [7, 18]). SCSe is NP-hard for strings of length 2 [18] and also NP-hard for an alphabet of size 2 [17]. LCSe is trivially solvable in polynomial time for strings of constant length, though NP-hard for an alphabet of size 2 [10].

Since it is difficult to find shortest supersequences (superstrings), it is desirable to have at least *minimal* supersequences (superstrings) that cannot be shortened by omitting some characters. A supersequence (superstring) S of a set L of strings is minimal if no proper

subsequence of S is also a supersequence (superstring) of L. Starting with any supersequence S of L a subsequence of S that is minimal can easily be found in polynomial time using the following strategy. Delete a character in S if the so-obtained subsequence of S is also a supersequence of L. Repeat this until no more characters can be deleted from the supersequence. The problem of finding a longest minimal supersequence is known to be MAX SNP-hard for strings over an alphabet of size 2 [14]. For strings of length 2 this problem is polynomial time solvable, whereas the complexity for strings of constant length $k \geq 3$ is open [4].

For superstrings the situation is different. A minimal superstring of L cannot necessarily be found by iterative deletions of single characters from some superstring as long as the string so obtained is required to be a superstring. As an example consider the superstring $S = abcbcd$ of $L = \{abc, bcd\}$. S is not minimal since deleting substring cb gives the superstring $abcd$ of L. But deleting only one character from S gives a string that is not a superstring of L. We identify new types of superstrings called "weak minimal" superstrings that can be obtained from a given superstring S in polynomial time.

More generally we study the complexity of problems which are a combination of a super-/subsequence problem with a super-/substring problem. Such problems have not been studied before. In particular, we focus on the following problems: given a string S and a set L of strings find a string that is a

1. minimal superstring (maximal substring) of L and subsequence of T,
2. minimal superstring (maximal substring) of L and supersequence of T,
3. minimal supersequence (maximal subsequence) of L and substring of T,
4. minimal supersequence (maximal subsequence) of L and superstring of T.

We show that some of these problems can be solved in polynomial time while others are NP-complete. E.g. we show that the complexity of finding a minimal supersequence S for a set of strings such that S contains a fixed string T as a substring depends on the string T — for some strings T the problem is NP-complete, whilst for others it is polynomial time solvable. We also consider some non-substring and non-super/non-subsequence variants, i.e. when we search for strings that do not have the given property. Moreover, we show that the Longest Minimal Non-Subsequence problem is MAX SNP-hard for strings over an alphabet of size 2, thus solving an open problem mentioned in [14]. Note, that the problem of finding a shortest supersequence of L that is a subsequence of T and the problem of finding a longest subsequence of L that is a supersequence of T have been studied in [14].

Let us give some motivating applications for our problems. The first two examples concern minimal superstring problems that are studied in Section 3. (1) In rockwool production a machine can produce rockwool of different diameters $D_1 < D_2 < \ldots < D_n$. Usually it is easy to change the production from smaller diameters to larger ones since the speed of production goes down. The other direction requires that the machine waits some time before it can speed up. Now assume we want to produce several lots of rockwool where every lot fills a truck. Each lot consists of a sequence of rockwool units of different diameters. To reduce costs for storing we assume that the rockwool for each lot is a substring in the production sequence of the machine. The production sequence is the sequence of different diameters that occur during production (e.g. the lot $D_5D_6D_2$ is a substring in the production sequence $D_3D_5D_6D_2D_4D_5$). To find a good production sequence we want to limit the number of switches from a larger to a smaller diameter. Hence we look for a minimal superstring S of the lot sequences that is a subsequence of $(D_1D_2 \ldots D_n)^k$. In this case S contains at most k switches from large to small diameter. (2) In other application scenarios similar to (4) one might be interested to have a large number of switches between two different products A and

B. In this case a minimal superstring that is a supersequence of one of the strings $(AB)^k$ or $(BA)^k$ is needed.

The next example is a maximal substring problem (see Section 4). (3) The webusage behaviour of a customer that visits the wepages of a company can be decribed by her navigation path through the webpages, i.e. by a string over the alphabet of the companies webpages (e.g. [12]). An interesting Data Mining problem is to find common usage patterns of customers. Assume one is interested to find for a set L of strings describing customers navigation paths a longest common substring that started possibly with page A then went on to the set of pages $\{B, C, D\}$ and ended in the set of pages $\{A, E\}$. Then the problem is to find a longest common substring of L that is a subsequence of the string $A(BCD)^k(AE)^l$ for a large enough $k, l > 0$.

The next two examples concern minimal supersequence problems as studied in Section 5. (4) Assume a conveyer belt consists of a sequence T of machines of different type. A product can be characterised by the sequence S of machines it has to pass during production process. The product can be produced on T if S is a subsequence of T. Assume that the conveyer belt T has to be extended on both sides such that a set L of new products can be produced on it and such that no machine is unnecessary. This is the problem of finding a string S that is a minimal supersequence of L and contains T as a substring. (5) Another problem is to find for a set of products characterised by a set L of strings a conveyer belt with no unnecessary machines and where a given sequence of neighboured machines is not allowed to be included (e.g. a cooling machine should not be placed after a heating machine). Then the problem is to find a minimal supersequence of L that does not contain a certain substring.

The organisation of the paper is as follows. Basic definitions are given in Section 2. Section 3 contains results about finding minimal superstrings. Maximal substring problems are studied in Section 4. Minimal supersequence and maximal subsequence problems are described in Section 5. Conclusions are given in Section 6.

2 Basic Definitions

A string over an alphabet Σ is a finite sequence of characters from Σ. For a string S the prefix of length k is denoted by $Pref_k(S)$. The empty string is denoted by λ. The concatenation of two strings S and T is denoted by ST. For S let $S^0 = \lambda$ and $S^i = SS^{i-1}$ for each integer $i \geq 1$. A string obtained from S by deleting zero or more characters is called a *subsequence* of S. T is a *supersequence* of S if S is a subsequence of T. A string S obtained from T by deleting a (possibly empty) prefix and a (possibly empty) suffix is called a *substring* of T. This is denoted by $S \overline{\ll} T$, and by $S \ll T$ if S is a *proper substring* of T (i.e. $S \overline{\ll} T$ and $S \neq T$). T is a *superstring* of S if S is a substring of T. S is a subsequence (supersequence, substring, superstring) of a set L of strings if S is a subsequence (supersequence, substring, superstring) of every string in L. The property that S is a substring of L is denoted by $S \overline{\ll} L$. S is a *non-supersequence* (*non-subsequence*, *non-substring*) of T if S is not a supersequence (subsequence, substring) of T. S is a non-supersequence (non-subsequence) of a set L of strings if S is a non-supersequence (non-subsequence) of every string in L. A supersequence (superstring, non-subsequence) S of a set of strings is *minimal* if no proper subsequence of S is a supersequence (superstring, non-subsequence). A subsequence (substring, non-supersequence) S of a set of strings is *maximal* if no proper supersequence of S is a subsequence (substring, non-supersequence).

Let $S = s_1 s_2 \dots s_l$ be a subsequence of $T = t_1 t_2 \dots t_k$. An *embedding* of S in T is a strictly increasing function $f : [1 : l] \to [1 : k]$ such that $s_i = t_{f(i)}$ for all $i \in [1 : l]$. We say that s_i *is mapped onto* $t_{f(i)}$ by f, $i \in [1 : l]$. An embedding of S into a set L of strings is a set $F = \{f_T \mid T \in L\}$ where for each $T \in L$ the function f_T is an embedding of S in T. An

embedding of L into S is a set $F = \{f_T \mid T \in L\}$ where for each $T \in L$ the function f_T is an embedding of T in S. A *run* of a string is a substring of maximal length that is of the form a^k for $a \in \Sigma$ and $k \geq 1$ (also called *a-run*).

3 Minimal Superstrings

In this section we consider the problem of finding, for a given string T and set L of strings, a subsequence (supersequence) S of T that is a minimal superstring of L. It is easy to see that the following characterisations are equivalent:

- S is a minimal supersequence of L.
- There does not exist a character s in S such that the string obtained after deleting s is also a supersequence of L.
- There does not exist a substring S' of S such that the string obtained after deleting S' is also a supersequence of L.
- For every embedding of L into S and every character s in S there exists at least one character of a string in L that is mapped onto s.

For minimal superstrings none of the analogous equivalences holds in general. Before we give examples showing this we define the corresponding types of minimal superstrings, i.e. the "weak minimal" superstrings as mentioned in the Introduction. A superstring S of a set L of strings is

- *embedding-minimal* (*e-minimal*) if for each embedding of L into S and for every character s of S the set of characters mapped onto s is not empty,
- *substring-deletion-minimal* (*sub-minimal*) if no proper subsequence of S, obtained by the deletion of exactly one substring, is a superstring,
- *character-deletion-minimal* (*char-minimal*) if no subsequence of S, obtained by deleting exactly one character, is a superstring,
- *prefix-suffix-deletion-minimal* (*pre-suf-minimal*) if no proper substring of S is a superstring.

Lemma 3.1. *(a) For a superstring S of a set L of strings the following implications hold: minimal $\Rightarrow$ sub-minimal $\Rightarrow$ char-minimal $\Rightarrow$ e-minimal $\Rightarrow$ pre-suf-minimal. (b) The inverse implication "char-minimal $\Rightarrow$ sub-minimal" holds for strings of length 2 but not for strings of length 3. The other inverse implications do not hold even for strings of length 2.*

Proof. (a) follows from the definitions. For (b), we give the following examples which deal with the inverse implications:

1. Let $L = \{ab, ac, bd, ca\}$ and $S = abcacbd$. It is not hard to show that S is sub-minimal. But S is not minimal since the proper subsequence $S' = acabd$ of S is also a superstring of L.

2. Let $L = \{cba, bac\}$ and $S = cbabac$. It is easy to show that S is char-minimal. But S is not sub-minimal since the subsequence $S' = cbac$ obtained by deleting the substring ba is a superstring of L.

3. Let $L = \{ba, ab\}$ and $S = baab$. S is e-minimal but not char-minimal since the subsequence $S' = bab$ obtained by deleting one a is a superstring of L.

4. Let $L = \{ab, c\}$ and $S = ab^ic$ for some integer $i \geq 2$. It is easy to show that S is pre-suf-minimal but not e-minimal.

It remains to show "char-minimal $\Rightarrow$ sub-minimal" holds for strings of length 2. For a contradiction assume there is a set L of strings of length 2 and a char-minimal superstring S that is not sub-minimal, i.e. $S = S'S''S'''$ for strings S', S'', S''', $|S''| \geq 2$ and $S'S''$ is also a superstring. Clearly, $S' \neq \lambda \neq S'''$. Consider the set of strings $L' \subset L$ that have at least one character in S'' in every embedding into S. Since S is char-minimal L' is not empty. It follows that in any embedding of L' into $S'S'''$, every string has to be embedded into the last character of S' — say a — and the first character of S''' — say b. Since S is char-minimal we derive that $|L'| = 1$ and $S'' = ab$. But then the string $S'aS'''$ is also a superstring of L and therefore S was not char-minimal. □

As mentioned in the Introduction, finding a shortest superstring of a set of strings is solvable in linear time for strings of length 2 and NP-complete for strings of length 3, and also for strings over a binary alphabet [5]. The complexity of finding any (not necessarily a shortest) minimal superstring is open for strings of length 3, and also for strings over an alphabet of size 2. Also, the complexity of finding a longest (sub-,char-,e-,pre-suf-)minimal superstring is not known. However, using a suffix tree (see Chapter 5 of [8] for an introduction to suffix trees) it can be checked in time $O(n|\Sigma|)$ whether a string of size $O(n)$ is a superstring of a set L of strings over an alphabet Σ, where n denotes the total length of all strings in L. This implies easily that finding just any sub-minimal superstring of a set of strings can be done in time $O(n^4|\Sigma|)$. Similarly, char-minimal and e-minimal superstrings can be found in time $O(n^3|\Sigma|)$. To find a pre-suf-minimal superstring start with any superstring S. Then using a suffix tree, find the rightmost occurrence in S of every string in L. Let S' be the shortest suffix of S that contains all rightmost occurrences of strings in L. Similarly find then the leftmost occurrence of every string of L in S'. Now the string S'' that is the shortest prefix of S' containing all leftmost occurrences of the strings in L is a pre-suf-minimal superstring of L. Hence, a pre-suf-minimal superstring can be found in time $O(n|\Sigma|)$.

We consider now the problem of finding a (sub-,char-,e-,pre-suf-)minimal superstring that can be of arbitrary length but has to be a supersequence (subsequence) of some given string.

Definition 3.2. Fixed Supersequence Minimal Common Superstring:
Given: A set L of strings and a string T over an alphabet Σ.
Question: Does there exist a minimal superstring of L which is a subsequence of T?

The problems Fixed Supersequence (sub-,char-,e-,pre-suf-)Minimal Common Superstring *are defined analogously. When the superstring does not have to be minimal the problem is called* Fixed Supersequence Common Superstring *problem.*

The Fixed Supersequence Common Superstring problem is polynomially equivalent to the Fixed Supersequence sub-Minimal Common Superstring problem. To see this, suppose that S is a superstring of L, and suppose that S' is a string obtained from S by deletion of a substring, such that S' is also a superstring of L. Then S' is also a subsequence of T. Analogously, this holds for the Fixed Supersequence char-Minimal (e-Minimal, pre-suf-Minimal) Common Superstring problems. In case of the Fixed Supersequence Minimal Common Superstring problem, such relationship is not immediate, since the complexity of the problem of deciding whether a superstring of a set L of strings is minimal is open.

Theorem 3.3. *The Fixed Supersequence (sub-,char-,e-,pre-suf-)Minimal Common Superstring problem and the Fixed Supersequence Common Superstring problem are NP-hard for strings of length 2.*

Proof. We reduce the 3-SAT ([LO2] in [6]) to our problem. Let a set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of clauses each of size three over a set $V = \{v_1, v_2, \ldots, v_n\}$ of variables be an instance of 3-SAT.

We define a set L of strings of length 2 and a string T over an alphabet Σ. For each variable v_h, $h \in [1:n]$ let $C_{i_1}, C_{i_2}, \ldots, C_{i_{p_h}}$ be the clauses with an unnegated occurrence of v_i and $C_{j_1}, C_{j_2}, \ldots, C_{j_{q_h}}$ be the clauses with an occurrence of $\bar{v}_i$ and define the corresponding strings $T_i = c_{i_1} c_{i_2} \ldots c_{i_{p_h}}$, $T'_i = c_{j_1} c_{j_2} \ldots c_{j_{q_h}}$, and $T''_i = v_i T_i \bar{v}_i v_i T'_i \bar{v}_1 \#_i$. Set $T = \#_0 T''_1 T''_2 \ldots T''_n$. Let $L = \{v_i \bar{v}_i \mid 1 \leq i \leq n\} \cup \{c_i \mid 1 \leq i \leq m\} \cup \{\#_i \mid 0 \leq i \leq n\}$.

Assume that S is a minimal superstring of L and a subsequence of T. Clearly, S contains every character $\#_i$, $1 \leq i \leq n$ exactly once. Since S contains the substring $v_i \bar{v}_i$ and is a subsequence of T the substring of S between $\#_{i-1}$ and $\#_i$ is in one of the following forms: i) $\#_{i-1} S_i v_i \bar{v}_i \#_i$ where S_i is a subsequence of T_i or ii) $\#_{i-1} v_i \bar{v}_1 S'_i \#_i$ where S'_i is a subsequence of T'_i. We obtain a truth assignment of V by setting v_i, $i \in [1:n]$ true if (i) holds for v_i and otherwise v_i is set false. The proof for the case that S is any superstring is very similar. To show that there exists a minimal superstring of L that is a subsequence of T when there exists a $\mathcal{C}$-satisfying truth assignment for V is easy. The proof for the case that S is any superstring is very similar, and the results for the Fixed Supersequence char-Minimal (e-Minimal, pre-suf-Minimal) Common Superstring problems follow since each is polynomially equivalent to the Fixed Supersequence Common Superstring problem. □

The fixed string T in the proof of Theorem 3.3 depends on the instance of 3-SAT. An interesting question is whether the problem remains NP-complete when T depends only on the size of the instance, i.e. does there exist an infinite sequence $T = s_1 s_2 \ldots$ over characters of some alphabet Σ such that the following problem is NP-complete: given a set of strings over Σ, $n \in \mathbb{N}$, is $Pref_n(T)$ a (sub-,char-,e-,pre-suf-)minimal superstring of L?

Corollary 3.4. *The Fixed Supersequence (sub-,char-,e-,pre-suf-)Minimal Common Superstring problem is NP-hard for strings over an alphabet of size* 2*.*

We omit the proof. The idea is to encode every character in the proof of Theorem 3.3 by a suitable string over an alphabet of size 2.

Definition 3.5. Fixed Subsequence Minimal Common Superstring:
Given: A set L of strings and a string T over an alphabet Σ.
Question: Does there exist a minimal superstring of L which is a supersequence of T?

The problems Fixed Subsequence (sub-,char-,e-,pre-suf-)Minimal Common Superstring *are defined analogously.*

Theorem 3.6. *The Fixed Subsequence (sub-,char-,e-)Minimal Common Superstring problem is NP-hard for strings of length 2.*

Proof. We reduce a 3-SAT version where for each variable the number of negated occurences equals the number of unnegated occurences and no variable occurs twice in a clause. Let a set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of clauses each of size three over a set $V = \{v_1, v_2, \ldots, v_n\}$ of variables be an instance of 3-SAT. Before we define a set L of strings of length 2 and a string T over an alphabet Σ we need some definitions and observations.

A string S' in L *supports* a character in a superstring S of L when S' can not be mapped into S without using this character. Clearly, every character in a (sub-,char-,e-)minimal superstring S of L has to be supported by some string in L. In the following definition of T each # stands for some character that occurs only once in T. We make the following observation: When T contains a substring of the form $\#ab\#$, $ab \in L$, and when the string ab is needed to support the substring ab of T (i.e. it is not possible that strings of the form ax and by support the a and the b) then the substring ab occurs only once in a (sub-,char-,e-)minimal superstring S of L with subsequence T and this occurence is between the corresponding # characters. A main tool in the proof is to use this observation to restrict the neighbourhoods of characters in parts of S as follows. We define a substring T_0 of T and a set $L_0 \subset L$ of strings so that

i) outside of the substring T_0 of S the character c_j, $j \in [1:m]$ can only have right neighbours c_j, x_j, or v_i^h when C_j is the hth clause with positive occurence of v_i and left neighbours c_j, y_j, or $\bar{v}_i^h$ when C_j is the hth clause with negated occurence of v_i,

ii) outside of the substring T_0 of S the character c_j, $j \in [m+1:(3/2)m]$ (without loss of generality we assume that m is even) can only have right neighbours c_j, x_j, or v_i, $i \in [1:n]$, and left neighbours c_j, y_j, or $\bar{v}_i$, $i \in [1:n]$.

For each variable v_i, $i \in [1:n]$ define $T_{v_i} = v_i^1\bar{v}_i^1v_i^2\bar{v}_i^2 \dots v_i^{n_i}\bar{v}_i^{n_i}v_i^{n_i+1}\#$ and let T_1 be the concatenation of all strings T_{v_i}, $i \in [1:n]$. For $j \in [1:(3/2)m]$ define $T_{C_j} = a_ja_jc_jc_jb_jb_j\#$. Let T_2 be the concatenation of all strings T_{C_j}, $j \in [1:(3/2)m]$. Set $T = T_0T_1T_2$. The set L contains the strings in L_0, all strings $\#$ that occur in T and the strings in the following sets $L_1 = \{v_i^h\bar{v}_i^h, \bar{v}_i^hv_i^{h+1} \mid h \in [1:n_i], i \in [1:n]\} \cup \{v_i^{n_i+1}v_i^1 \mid i \in [1:n]\}$, $L_2 = \{x_iy_i \mid i \in [m+1:(3/2)m]\}$, $L_3 = \{a_ia_i, b_ib_i, c_ic_i \mid i \in [1:(3/2)m]\}\}$. Now we show that a $\mathcal{C}$-satisfying truth assignment of V exists when there exists a (sub-,char-,e-)minimal superstring of L that contains T as a subsequence.

Let S be a (sub-,char-,e-)minimal superstring of L with subsequence T. Consider an embedding of L in S. The following facts are easy to show: i) every character $\#$ can occur only once in S and the corresponding string $\#$ supports it, ii) every of the substrings a_ia_i, b_ib_i, c_ic_i, $i \in [1:(3/2)m]$ in substring T of S has to be supported by the corresponding string in $L_3 \subset L$, iii) of the $(7/2)m+n$ strings in $L_1 \cup L_2$ exactly $3m$ have to occur in S as a neighbour to a character c_j, $j \in [1:(3/2)m]$ in the subsequence T in order guaranty that none of c_j's neighbours a_j and b_j in T is its neighbour in S, $j \in [1:(3/2)m]$, iv) $(1/2)m+n$ strings in L_1 are needed to support the (non-$\#$)characters in T_{v_i}, $i \in [1:n]$.

It follows that exactly n_i+1 of the strings in L_1 have to support the (non-$\#$)characters in T_{v_i}, $i \in [1:n]$. There are only two possibilities: Either all strings $\{v_i^h\bar{v}_i^h \mid h \in [1:n_i]\} \cup \{v_i^{n_i+1}v_i^1\}$ or all strings $\{\bar{v}_i^hv_i^{h+1} \mid h \in [1:n_i]\} \cup \{v_i^{n_i+1}v_i^1\}$ support the (non-$\#$)characters in T_{v_i}. This allows to define a truth assignment for the variables in V. In the first case v_i is set false and in the second case v_i is set true.

For each substring c_jc_j, $j \in [1:m]$ only the string x_jy_j and strings in L_2 can guaranty that (i) is satisfied when they occur as neighbours in S. Hence at least one string from L_2 must occur in S as neighbour of c_j. This is possible for a string $v_i^h\bar{v}_i^h$ only when C_j is the hth clause that contains v_i and v_i is true (i.e., the string $v_i^h\bar{v}_i^h$ is not needed to support characters in T_{v_i}) or for a string $\bar{v}_i^hv_i^h$ when C_j is the hth clause that contains $\bar{v}_i$ and v_i is false (i.e., the string $\bar{v}_i^hv_i^h$ is not needed to support characters in T_{v_i}). Hence there must be at least one true literal in each clause. The other direction of the proof is easy. □

By encoding every character in the proof of Theorem 3.6 by a suitable string over an alphabet of size 2 we obtain the following corollary.

Corollary 3.7. *The Fixed Subsequence (sub-,char-,e-)Minimal Common Superstring problem is NP-hard for strings over an alphabet of size* 2.

By contrast with the previous result, Fixed Subsequence pre-suf-Minimal Common Superstring is solvable in linear time, as we show now.

Theorem 3.8. *The Fixed Subsequence pre-suf-Minimal Common Superstring problem is solvable in time $O(n)$ where n is the total length of the input strings.*

Proof. Let a set L of strings and a string T over an alphabet Σ be given. Let m be the maximal length of a run of a string in L. For a string S let $\alpha(S)$ be the string that is obtained from S by shortening the leftmost run and the rightmost run so that each has length one. Let $\alpha(L) = \{\alpha(S) \mid S \in L\}$. For a string S let $S\triangleright$ be the string that is

Type of minimality	any	shortest	longest	fixed supersequence	fixed subsequence
minimal	? $O(n)$: $l = 2$ [5]	NPh: $\|\Sigma\| = 2$ $l = 3$	?	NPh: $l = 2$, Th. 3.3 $\|\Sigma\| = 2$, Cor. 3.4	NPh: $l = 2$, Th. 3.6 $\|\Sigma\| = 2$, Cor. 3.7
sub-minimal	$O(n^4\|\Sigma\|)$		?		
char-minimal	$O(n^3\|\Sigma\|)$	$O(n)$:	?		
e-minimal		$l = 2$	?		
pre-suf-min.	$O(n\|\Sigma\|)$	[5]	$\not\exists$		$O(n)$, Th. 3.8
none					

Table 1: Complexity of minimal superstring problems: P=polynomial time, NPh=NP-hard, $\not\exists$=may not exist, l=length of longest string in L, n=total length of strings in L, Σ=alphabet

obtained from S by doubling the rightmost character until the length of the rightmost run is $m+1$. Strings $\triangleleft S$ and $\triangleleft S \triangleright$ are defined analogously by doubling the leftmost (respectively the leftmost and the rightmost) character of S until the length of the leftmost (respectively the leftmost and the rightmost) run is $m+1$. We consider three cases.

Case 1: $\alpha(L)$ contains at least two strings with ≥ 3 runs. Let $T_1, T_2, \ldots, T_m$ be the strings in $\alpha(L)$ with ≥ 3 runs, $m \geq 2$. Let S' be the string in L with $\alpha(S') = T_1$ that has the longest leftmost run and S'' be the string in L with $\alpha(S'') = T_m$ that has the longest rightmost run. Let S''' be a string that has only runs of length $m+1$, contains T as a subsequence, and is a superstring of every string $S \in L$ with at most two runs. Then it is easy to show that the string $S' \triangleright S''' \triangleleft T_2 \triangleright \triangleleft T_3 \triangleright \ldots \triangleleft T_{m-1} \triangleright \triangleleft S''$ is a pre-suf-minimal superstring of L that contains T as a subsequence.

Case 2: Case 1 does not hold and there are at least two strings in L with a run of length ≥ 2. First assume that $\alpha(L)$ contains one string T_1 with ≥ 3 runs and it has a run of length ≥ 2. Then $T_1 = a^{n_1}b^{n_2}c^{n_3}$, $a, b, c \in \Sigma$, $a \neq b \neq c$, for some integers n_1, n_2, n_3 at least one of them ≥ 2. Let S' be the shortest string that contains all strings $S \in L$ with $\alpha(S) = T_1$ and also every string of the form $a^k b^l$ with integers k, l, $l \leq n_2$ as a substring. Further, assume there exists a string in L that is not substring of S' and has a run of length ≥ 2. Then a string S'' can be found in linear time that is a suf-minimal superstring of the remaining strings in L where the first run of S'' has length one and the string that supports the rightmost character has a run of length ≥ 2. Let S''' be a string that has only runs of length 1, contains T as a subsequence, and the last character of S''' is different from the first character of S''. Then the string $S'S'''S''$ is a pre-suf-minimal superstring of L that contains T as a subsequence. Most remaining subcases can be proved similarly (a few subcases can be detected where there does not exist a pre-suf-minimal superstring of L with subsequence T).

Case 3: Neither of Cases 1 and 2 holds. Proof omitted since it can be shown with similar techniques as the other cases. □

Table 1 summarises the complexity results of this section.

4 Maximal Substrings

In this section we consider finding maximal substrings of a set of strings. In analogy to Section 3, maximal substring problems with a fixed supersequence (subsequence) are defined. Firstly, we define certain types of "weak maximal" substrings. A substring S of a set L of strings is

– *substring-insertion-maximal* (*sub-maximal*) if no proper supersequence of S, that can be obtained by the insertion of exactly one string, is a substring,

– *character-insertion-maximal* (*char-maximal*) if no supersequence of S that can be obtained by inserting exactly one character is a substring,

– *prefix-suffix-insertion-maximal* (*pre-suf-maximal*) if no proper superstring of S is a substring.

Lemma 4.1. *For a substring S of a set L of strings the following implications hold: maximal $\Rightarrow$ sub-maximal $\Rightarrow$ char-maximal $\Rightarrow$ pre-suf-maximal and the inverse implications do not hold in general.*

Proof. The implications can be shown easily, the following examples show that the inverse implications do not hold. (1) Let $L = \{aabacacb, acacbaab\}$ and $S = aab$. S is sub-maximal but not maximal since the string $acacb$ is also a substring. (2) Let $L = \{accbab, abaccb\}$ and $S = ab$. S is char-maximal but not sub-maximal since the string $accb$ obtained by inserting cc into S is also a substring. (3) Let $L = \{abaa, aaba\}$ and $S = aa$. Clearly, S is pre-suf-maximal but not char-maximal since aba is also a substring. □

Definition 4.2. Fixed Subsequence Maximal Common Substring:
Given: A set L of strings and a string T over an alphabet Σ.
Question: Does there exist a maximal substring of L which is a supersequence of T?

The problems Fixed Subsequence (sub-,char-,pre-suf-)Maximal Common Substring *and* Fixed Supersequence (sub-,char-,pre-suf-) Maximal Common Substring *are defined analogously.*

The Longest Common Substring problem is linear-time solvable: Hui [7] has shown that, for a set L containing k strings, a longest substring of L may be found in $O(n)$ time, where n denotes the *total* length of all strings in L. Hui's approach involves the use of suffix trees and *lowest common ancestors* (see Chapter 8 of [8] for a definition of lowest common ancestors and a description of how they may be computed efficiently). Gusfield [8, §9.7] presents an in-depth description of Hui's method, demonstrating how a simpler $O(kn)$ algorithm for the problem [8, §7.6] may be refined in order to achieve the $O(n)$ bound. In this section we show that *all* pre-suf-maximal substrings of L may be found in $O(n)$ time. As a corollary we obtain that a shortest pre-suf-maximal substring of L may be found in $O(n)$ time (note that this problem is a minimaximal optimisation problem with a special partial order property as studied in [11]).

Our algorithm makes use of Hui's methods for solving the Longest Common Substring problem. Additionally, some aspects of our approach bear similarities to the algorithm for finding all (pre-suf-) maximal repeats in a string in $O(n)$ time [8, §7.12.1]. However our task here involves k strings; nevertheless, by using suitable data structures, we achieve time bound $O(n)$ for our problem. We leave open whether the corresponding problems for (sub-,char-)maximal substrings can be solved in linear time (polynomial time is trivial).

We firstly establish some definitions relating to suffix trees (the terminology follows that of Gusfield [8, §5.2]). Let $\mathcal{T}$ be a suffix tree for a string S. The *label of a path* from the root of $\mathcal{T}$ to a node v is the concatenation, in order, of the substrings labelling the edges of that path. The *path label* of a node v is the label of the path from the root of $\mathcal{T}$ to v. For any node v, the *string depth* of v is the number of characters in the path label of v.

Theorem 4.3. *For a set L of k strings with total length n all pre-suf-maximal substrings of L can be found in $O(n)$ time.*

Proof. Suppose that $L = \{S_i : 1 \leq i \leq k\}$. To each string S_i $(1 \leq i \leq k)$, we append a unique termination symbol $\$_i$ not occurring in Σ; let S'_i be the resultant string and S the concatenation of the strings S'_i $(1 \leq i \leq k)$. Now suppose that X is a suffix of S, beginning at position i of S (i.e. i is the *suffix position of X in S*). This position of S corresponds to a unique string S'_j for some j $(1 \leq j \leq k)$. We call j the *string identifier of X in S*. Define the *left character of X in S* to be the $(i-1)$th character of S if $i > 1$, or $\$_0$ (a symbol not occurring in Σ) if $i = 1$.

We build the suffix tree $\mathcal{T}$ for the string S, storing two pieces of information at each leaf node. Recall that each leaf node v of $\mathcal{T}$ corresponds to a unique suffix X of S. Define the *string identifier of v in S* to be the string identifier of X in S, and define the *left character of v in S* similarly. Store both of these values at v. (For the purposes of this algorithm, it is not necessary to store at v the suffix position of X in S.) It is clear that the construction of this paragraph may be carried out in $O(n)$ time, which is the time required to build a suffix tree [2].

For a node v of $\mathcal{T}$, let $\mathcal{T}_v$ denote the subtree of $\mathcal{T}$ with root v. Let $C(v)$ denote the number of distinct string identifiers that appear at the leaves of $\mathcal{T}_v$. Define a *matching node* of $\mathcal{T}$ to be an internal node v of $\mathcal{T}$ such that $C(v) = k$. It follows that a string P is a substring of L if and only if P is, or is a prefix of, the path label of some matching node v of $\mathcal{T}$. Thus a string P is a maximal substring of L implies that P is the path label of some matching node v of $\mathcal{T}$. Computing $C(v)$ for each node v may be carried out in $O(n)$ time overall [7]. We now consider *R-tight matching nodes*. Such a node v is a matching node such that no child of v in $\mathcal{T}$ is a matching node.

Claim 4.4. *Let v be a matching node of $\mathcal{T}$ and let P be the path label of v. Then v is an R-tight matching node if and only if $P\sigma \not\ll L$, for any $\sigma \in \Sigma$.*

Proof of Claim 4.4. Suppose that v is an R-tight matching node and $Q = P\sigma \ll L$, for some $\sigma \in \Sigma$. Then $Q \ll S_j$ for each j $(1 \leq j \leq k)$, which implies that there are k suffixes X_j $(1 \leq j \leq k)$ of S such that, for each j $(1 \leq j \leq k)$, X_j has string identifier j in S, and Q is a prefix of X_j. Thus, by definition of $\mathcal{T}$, there is a matching node w in $\mathcal{T}_v$, where w is a child of v (w has path label R, such that either $Q = R$, or Q is a prefix of R). Thus v has a child that is a matching node, a contradiction. Conversely, suppose that $P\sigma \not\ll L$ for each $\sigma \in \Sigma$, and v is not an R-tight matching node. Then v has a child w that is a matching node; let Q be the path label of w. Then by definition of $\mathcal{T}$, there is some $\sigma \in \Sigma$ such that $P\sigma$ is a prefix of Q (possibly $P\sigma = Q$). Thus we reach a contradiction, since $Q \ll L$ implies that $P\sigma \ll L$. □

It is clear that the R-tight matching nodes may be determined by a straightforward traversal of $\mathcal{T}$, in $O(n)$ time, once the $C(v)$ values have been computed. By Claim 4.4, the path label of an R-tight matching node is a substring of L that cannot be extended to the right to give another common substring of L. Next, we show how to locate substrings that cannot be extended to the left, in addition to being non-extendible to the right. For an R-tight matching node v of $\mathcal{T}$ and for any $\sigma \in \Sigma$, let $D_\sigma(v)$ denote the number of distinct string identifiers among all leaves of $\mathcal{T}_v$ with left character σ in S. Define an *LR-tight matching node* v to be an R-tight matching node v such that, for all $\sigma \in \Sigma$, $D_\sigma(v) < k$.

Claim 4.5. *Let v be an R-tight matching node of $\mathcal{T}$ and P be the path label of v. Then v is an LR-tight matching node if and only if $\sigma P \not\ll L$, for any $\sigma \in \Sigma$.*

Proof of Claim 4.5. Suppose that v is an LR-tight matching node and $Q = \sigma P \ll L$, for some $\sigma \in \Sigma$. Then $Q \ll S_j$ for each j $(1 \leq j \leq k)$, which implies that there are k suffixes X_j $(1 \leq j \leq k)$ of S such that, for each j $(1 \leq j \leq k)$, P is a prefix of X_j, X_j has string identifier j in S, and σ is the left character of X_j in S. Thus, by construction of $\mathcal{T}$, we

```
for each leaf node w of T_v loop
   let σ be the left character of w;
   if σ ∈ Σ then
      add w to S_{σ,v};
   end if;
end loop;
for each nonempty set S_{σ,v} loop
   D_σ(v) := 0;
   for each w ∈ S_{σ,v} loop
      let t be the string identifier of w;
      if not visited[t] then
         visited[t] := true;
         D_σ(v) := D_σ(v) + 1;
         if D_σ(v) = k then
            halt;     {v is not LR-tight}
         end if;
      end if;
   end loop;
   for each w ∈ S_{σ,v} loop
      let t be the string identifier of w;
      visited[t] := false;
   end loop;
end loop;
{v is LR-tight}
```

Figure 1: An algorithm for deciding whether v is an LR-tight matching node, given that v is an R-tight matching node of $\mathcal{T}$.

have that $\mathcal{T}_v$ has a leaf node with left character σ and string identifier j in S, for each j $(1 \leq j \leq k)$. Hence $D_\sigma(v) = k$, a contradiction. Conversely, suppose that $\sigma P \not\ll L$ for each $\sigma \in \Sigma$, and v is not an LR-tight matching node. Then there is some $\sigma \in \Sigma$ such that $\mathcal{T}_v$ has a leaf node with left character σ and string identifier j in L, for each j $(1 \leq j \leq k)$. Hence $Q = \sigma P$ satisfies $Q \lll S_j$, for each j $(1 \leq j \leq k)$, so that $Q \lll L$, a contradiction. □

By Claim 4.5, a string P is a maximal substring of L if and only if P is the path label of an LR-tight matching node v of $\mathcal{T}$. We now show how to efficiently determine the R-tight matching nodes that are LR-tight matching nodes.

Claim 4.6. *The LR-tight matching nodes in $\mathcal{T}$ may be found in $O(n)$ time.*

Proof of Claim 4.6. An algorithm for deciding whether a given R-tight matching node v of T is LR-tight is shown in Figure 1. The algorithm calculates only those $D_\sigma(v)$ values for which $\mathcal{T}_v$ contains at least one leaf node with left character $\sigma \in \Sigma$ (otherwise we may assume that $D_\sigma(v) = 0$); once such a value has been computed, if $D_\sigma(v) = k$ then v cannot be an LR-tight matching node. If this is not the case after all such computations, v is LR-tight.

To begin, the algorithm makes one pass over the leaf nodes in $\mathcal{T}_v$, constructing the sets $S_{\sigma,v}$, where $\sigma \in \Sigma$. Initially we assume that $S_{\sigma,v} = \emptyset$, and at the termination of the first for loop, each $S_{\sigma,v}$ contains the leaf nodes in $\mathcal{T}_v$ having left character σ. Clearly, with suitable data structures, these sets may be constructed in $O(r)$ overall time (avoiding the explicit initialisations $S_{\sigma,v} := \emptyset$), and they use $O(r)$ total space, where r is the number of leaf nodes in $\mathcal{T}_v$.

The second for loop considers in turn each nonempty set $S_{\sigma,v}$. For each, the number of distinct string identifiers among leaf nodes of $\mathcal{T}_v$ having left character σ is computed. This is done with the aid of a boolean array *visited*, having k entries. We assume that every entry

visited is initialised to false at the very outset (i.e. before the algorithm is invoked on any R-tight matching node), and once $D_{\sigma,v}$ has been computed, those values of *visited* that were altered are reset to false. Clearly the second for loop may be implemented to run in $O(r)$ overall time and $O(k+r)$ total space.

Given that distinct R-tight matching nodes contain disjoint sets of leaf nodes, it follows that the overall time and space used by the algorithm of Figure 1 for all R-tight matching nodes is $O(n)$. □

Thus Claim 4.6 implies that all pre-suf-maximal substrings of L may be identified in $O(n)$ time by representing each such substring α as a pair (i, j), so that α comprises all characters of S between positions i and j inclusive (assuming that, in practice, each edge label β of $\mathcal{T}$ is represented similarly by a pair of indices (k, l) [8, p.104]). □

Corollary 4.7. *A shortest (sub-,char-)maximal substring of a set of strings L can be found in time $O(n+l^3)$, where l is the length of the shortest string, and a shortest pre-suf-maximal substring can be found in time $O(n)$.*

Proof. Consider the suffix tree $\mathcal{T}$ defined in the proof of Theorem 4.3. Having marked all the LR-tight matching nodes of $\mathcal{T}$, a final traversal of the tree will establish an LR-tight matching node v of smallest string depth, in $O(n)$ time. By construction, the path label P of v corresponds to a shortest pre-suf-maximal substring of L. Thus the overall time complexity of this algorithm is $O(n)$. To find a shortest maximal substring it is enough to find a shortest pre-suf-maximal substring that is not subsequence of some other pre-suf-maximal substring. Since there are at most l pre-suf-maximal substrings this can be done in time $O(l^3)$ and the time bound of the corollary follows. Similarly, shortest (sub-,char-)maximal substrings can be found. □

Corollary 4.8. *Let L be a set of strings, let T be a string, and let l be the length of the shortest string in L. Then*

1. *The following time bounds hold for the Fixed Supersequence (sub-,char-,pre-suf-)Maximal Common Substring problems: i) (sub-,char-) maximal: $O(n + l \cdot |T| + l^3)$, ii) pre-suf-maximal: $O(n + l \cdot |T|)$.*

2. *The following time bounds hold for the Fixed Subsequence (sub-,char-,pre-suf-)Maximal Common Substring problems: i) (sub-,char-) maximal: $O(n+l^3)$, ii) pre-suf-maximal: $O(n + l^2)$.*

Proof. The proofs of Theorem 4.3 and Corollary 4.7 show how all maximal, sub-maximal, char-maximal, and pre-suf-maximal substrings of L can be found. In each case there are $O(l)$ such substrings, and each substring can be tested as to whether it is a subsequence of T in $O(|T|)$ time. □

Table 2 summarizes the complexity results of this section.

5 Minimal Supersequences, Maximal Subsequences

The problem of finding for a given set of strings, a shortest (respectively longest) of the following sequences was studied by several authors: minimal supersequence, maximal non-supersequence, maximal subsequence, or minimal non-subsequence [4, 9, 10, 13, 17, 19]. It is known that each of these problems is NP-hard. Moreover, it is known that finding a longest minimal supersequence, shortest maximal subsequence and shortest maximal non-supersequence are MAX SNP-hard over an alphabet of size 2, i.e. there does not exist

Type of maximality	shortest	longest	fixed supersequence T	fixed subsequence T
maximal	$O(n+l^3)$	$O(n)$ [7]	$O(n+l\cdot\|T\|+l^3)$	$O(n+l^3)$
sub-maximal				
char-maximal				
pre-suf-maximal	$O(n)$		$O(n+l\cdot\|T\|)$	$O(n+l^2)$

Table 2: Complexity of maximal substring problems: l=length of shortest string in L, n=total length of strings in L, Σ=alphabet

a polynomial time approximation scheme for these problems unless P=NP [14]. In the following we show that finding a longest minimal non-subsequence is MAX SNP-hard over an alphabet of size 2. It remains open as to whether each of the problems of finding a shortest supersequence, longest subsequence, shortest non-subsequence and a longest non-supersequence is MAX SNP-hard over an alphabet of constant size (when it exists). In addition, we consider the problem of finding for a set of strings maximal and minimal (non-)super- and (non-)subsequences of arbitrary length that contain (or do not contain) a given string as a substring. For most of these problems we show NP-hardness even over an alphabet of size 2.

Theorem 5.1. *Given a set L of strings over an alphabet of size 2 it is MAX SNP-hard to find a longest minimal non-subsequence of L.*

Proof. We only sketch the proof, as it is somewhat similar to the proofs of the MAX SNP-hardness results in [14]. We L-reduce the Independent and Dominating Set-B problem to our problem (see [14]). The problem is to find for a graph $G=(V,E)$ with maximum degree B a smallest vertex set $V'\subset V$, $|V'|\leq k$ such that for every $v\in V-V'$ there exists $w\in V'$ with $\{v,w\}\in E$ (i.e., V' is a dominating set) and for all $u,v\in V'$, $\{u,v\}\notin E$ (i.e., V' is an independent set). Define a set L of strings over $\{0,1\}$: for $i\in[0:n]$ let $S_i=(10)^i11(01)^{n-i}$. For each edge $e_l=\{v_i,v_j\}\in E$, $i<j$, $l\in[1:m]$ let $T_l=(01)^{i-1}0(01)^{j-i-1}0(01)^{n-j}0$. Set $L=\{S_i\mid i\in[0:n]\}\cup\{T_1,T_2,\ldots,T_m\}$.

No minimal non-subsequence of L contains $\geq n+3$ zeros since every string in L contains $\leq n+1$ zeros. Clearly, every minimal non-subsequence of L containing $n+2$ zeros contains no one and therefore has length $n+2$. Every minimal non-subsequence S of L has $\leq n+3$ ones since every string in L has $\leq n+2$ ones. If S has exactly $n+3$ ones then $S=1^{n+3}$. It can be shown for $k\leq\frac{n}{2}-3$ that there exists a minimal non-subsequence of L with length $\geq 2n-k+1\geq n+\frac{n}{2}+4>n+3$ if and only if there exists an independent dominating set of size $\leq k$ for G. Further, L has an optimal solution with length $\leq 2n+1-opt(G)\leq(2B+3)opt(G)$ where $opt(G)$ is the size of the optimal solution for G. Hence we have an L-reduction. □

Definition 5.2. Fixed Substring Minimal Common Supersequence:
Given: A set L of strings and a string T over an alphabet Σ.
Question: Does there exist a minimal supersequence of L which contains T as a substring?

Fixed Substring Maximal Common Non-Supersequence, Fixed Substring Maximal Common Subsequence, Fixed Substring Minimal Common Non-Subsequence, Fixed Superstring Minimal Common Supersequence *and* Fixed Superstring Maximal Common Subsequence *are defined analogously.*

Clearly, Fixed Superstring Minimal Common Supersequence and Fixed Superstring Maximal Common Subsequence are polynomial-time solvable. For the other problems, we need the following theorem shown in [14].

Theorem 5.3. *Given a set L of strings over a binary alphabet and integers k_0, k_1. The following problems are NP-complete: Find a string containing exactly k_0 zeros and k_1 ones that is a (1) supersequence of L ((k_0,k_1)-Super), (2) subsequence of L ((k_0,k_1)-Sub), (3) non-supersequence of L ((k_0,k_1)-Non-Super), (4) non-subsequence of L ((k_0,k_1)-Non-Sub).*

The proof of Theorem 5.3 in [14] shows that the problems remain NP-complete in the following special cases (that we use in this section): (1) (k_0,k_1)-Super: each 1-run of a string in L has length 1, each string in L contains k_0-1 zeros and ends with a zero, (2) (k_0,k_1)-Sub: each 1-run of a string in L has length 1.

Theorem 5.4. *The following problems are NP-complete over an alphabet of size 2 even if the given non-substring T has constant length: (a) Fixed Substring Minimal Common Supersequence, (b) Fixed Substring Maximal Common Subsequence, (c) Fixed Substring Maximal Common Non-Supersequence, (d) Fixed Substring Minimal Common Non-Subsequence.*

Proof. We prove only (a) and (b). Results (c) and (d) can be proved by reductions from (k_0,k_1)-Non-Supersequence and (k_0,k_1)-Non-Subsequence. To prove (a) we reduce (k_0,k_1)-Supersequence. Let the set $L^* = \{T_l^* \mid l \in [1:m]\}$ of strings be an instance of (k_0,k_1)-Super where each 1-run has length 1, each string contains k_0-1 zeros and ends with a zero. We define a set L of strings over $\{0,1\}$ and a string T as follows. Let $T = 1100011$ and $L = L' \cup \{S_0, S_1\}$ with $L' = \{T_l^*1101 \mid T_i \in L^*\}$, $S_0 = 0^{k_0}011$, $S_1 = (10)^{k_1}1000$. It is now easy to verify that there exists a supersequence S^* for L^* with k_0 zeros and k_1 ones iff there exists a minimal supersequence S of L with substring T.

In fact, let S be a minimal supersequence of L with substring T. Since S is supersequence of L it must be of the form $S'0S''$, where S'' contains the subsequence 1101 and $S'0$ is a supersequence of L^*. Clearly S' has at least k_0-2 zeros. Since the prefixes of the strings in L that might have to be embedded in S' (i.e., $L^* \cup Pref_{k_0}(S_0) \cup Pref_{2k_1+2}(S_1)$) have only 1-runs of length 1 and S is minimal, it must be that S' has only 1-runs of length one. Hence, the substring T of S can only be embedded in the suffix S'' of S. Since S_0 is the only string that can support a second 1-run of length two in S'' there can be at most k_0-1 zeros in S'. Since S_1 is the only string that can support a 0-run of length three in S'' there can be at most k_1 zeros in S'. Hence $S'0$ is a supersequence of L^* with at most k_1 ones and at most k_0 zeros.

Vice versa, let S^* be a supersequence of L^* that contains exactly k_0 zeros and k_1 ones; we can assume that S^* has only 1-runs of length 1, since all strings in L^* have this property. Then $S = S^*1100011$ is a minimal supersequence for L with substring T. Indeed, to embed L' in S, the first 1-run of length two in the suffix 1100011 of S is required. To embed S_0 in S, the second 1-run of length two in the suffix 1100011 of S is required. To embed S_1 in S, the 0-run of length three in the suffix 1100011 of S is required. Moreover, to embed S_0 and S_1 in S every zero and every one in S^* is required.

(b) We reduce (k_0,k_1)-Subsequence. Let strings T_l^*, $l \in [1:m]$ be an instance of (k_0,k_1)-Sub where each 1-run has length 1. We define a set L of strings over the alphabet $\{0,1\}$ and a string T as follows. Let $T = 011011$, $S_0 = (10)^{k_0}10111011$, and $S_1 = 0^{k_0}(10^{k_0})^{k_1}0110111$. For each string T_l^*, $l \in [1:m]$ let $T_l = T_l^*01110111$. Set $L = \{S_0, S_1\} \cup \{T_1, T_2, \ldots, T_m\}$. It is shown in the following that there exists a subsequence with k_0 zeros and k_1 ones of $\{T_l^* \mid l \in [1:m]\}$ iff there exists a maximal subsequence of L that contains the substring T.

Let S be a maximal subsequence of L with substring T. Consider an embedding of S into L. Due to the maximality of S, for every 1-run of length 2 in T there must be a string S' in L such that both ones of the 1-run are mapped onto an 1-run in S' of length ≥ 2. Otherwise a zero could be inserted between the ones. The maximality implies that this is possible only when T is a suffix of S, i.e. S is of the form $S = T^*T$ for some string T^*. The rightmost zero of T in S must be mapped onto the rightmost zero of S_0. Otherwise, in each string of L

there would be the subsequence 111 to the right of the zeros onto which the rightmost zero of T in S is mapped. Similarly it follows that, both ones of the run 11 of T in S are mapped onto the second (seen from the right) 1-run of S_1. Due to the maximality of S it follows that

> $(*)$ the left zero of T in S has to be mapped onto the second (seen from the right) zero in S_0 and onto the second zero in S_1.

Therefore T^* is a subsequence of the strings T_l^*, $l \in [1:m]$ that contains exactly k_0 zeros (this is clear since S_0 contains only k_0 zeros to the left of the second rightmost zero and since $(*)$ would not be satisfied if T^* contains $< k_0$ zeros). Moreover, T^* contains exactly k_1 ones (this is clear since S_1 contains only k_1 ones to the left of the second rightmost zero and since $(*)$ would not be satisfied if T^* contains $< k_1$ ones).

On the other hand assume that there exists a subsequence T^* of $\{T_l^* \mid l \in [1:m]\}$ with k_0 zeros and k_1 ones. Clearly every 1-run of T^* has length one. Then T^* is a subsequence of the prefix $(10)^{k_0}1$ of S_0 and a subsequence of the prefix $0^{k_0}(10^{k_0})^{k_1}$ of S_1. In every embedding of T^* into the prefix $(10)^{k_0}1$ of S_0 the rightmost character of T^* can not be embedded to the left of the rightmost 0-run. Similarly, in every embedding of T^* into the prefix $0^{k_0}(10^{k_0})^{k_1}$ of S_1 the rightmost character of T^* can not be embedded to the left of the rightmost 0-run. It is easy to show that $S = T^*T$ is a maximal subsequence of L. □

Remarks: (1) Similar proofs show NP-completeness for the following problems:

(a) Find a minimal supersequence that contains two 1-runs of length 2 and one 0-run of length 3.

(b) Find a maximal subsequence that contains two 1-runs of length 2.

(c) Find a maximal non-supersequence that contains a 1-run of length 2.

(d) Find a minimal non-subsequence that contains a 1-run of length 3 and a 1-run of length 5.

(2) Fixed Substring Minimal Common Supersequence, Fixed Substring Maximal Common Subsequence, Fixed Substring Maximal Common Non-Supersequence, and Fixed Substring Minimal Common Non-Subsequence are NP-complete for strings over $\{0,1\}$ when one requires additionally that the string T is a suffix (or prefix) of constant length of the minimal supersequence (maximal subsequence, maximal non-supersequnce, minimal non-subsequence) S.

Definition 5.5. Fixed Non-Substring Minimal Common Supersequence:
Given: A set L of strings and a string T over an alphabet Σ.
Question: Does there exist a minimal supersequence of L that does not contain T as a substring?

Fixed Non-Substring Maximal Common Non-Supersequence, Fixed Non-Substring Maximal Common Subsequence, *and* Fixed Non-Substring Minimal Common Non-Subsequence *are defined analogously.*

Theorem 5.6. *The following problems are NP-complete over an alphabet of size 2 even if the given non-substring T has constant length: (a) Fixed Non-Substring Minimal Common Supersequence, (b) Fixed Non-Substring Maximal Common Subsequence, (c) Fixed Non-Substring Maximal Common Non-Supersequence.*

Proof. We prove only (a); (b) and (c) can be proven by reduction from (k_0, k_1)-Subsequence. To prove (a) we reduce (k_0, k_1)-Supersequence. Let the set $L^* = \{T_l^* \mid l \in [1 : m]\}$ of strings be an instance of (k_0, k_1)-Super where each 1-run has length 1. We define a set L of strings over $\{0, 1\}$ and a string T as follows. Let $T = 11$ and $L = L' \cup \{S_0, S_1\}$ with $L' = \{T_l^* 11011\}$, $S_0 = 0^{k_0}0111$, $S_1 = 1^{k_1}1110$, $S_2 = 0^{k_0}11011$, and $S_3 = 1^{k_1}11011$. We show that there exists a supersequence with k_0 zeros and k_1 ones of L^* iff there exists a minimal supersequence of L that does not contain the substring T.

Let S be a minimal supersequence of L that does not contain substring T. Since S is a supersequence of $L' \cup S_2, S_3$ it has a subsequence of the form $S'11011$ where S' is a supersequence of L^* and contains at least k_0 zeros and k_1 ones. Since S does not contain T there must be at least two additional zeros between the 1-runs of length two in $S'11011$. Since S_3 contains only one zero the other zero must be supported by S_2. But then can not contain $\geq k_0 + 1$ zeros because otherwise S_2 can be embedded in $S'11011$. Similarly, since S_2 contains only $k_0 + 1$ zeros and k_0 of them can can be embedded into S' it follows that one additional zero must be supported by S_3. This not possible when S' contains $\geq k_1 + 1$ zeros. Thus, S' is a subersquence of L^* that contains exactly k_0 zeros and k_1 ones.

Vice versa, let S^* be a supersequence of L^* that contains exactly k_0 zeros and k_1 ones. Since every string in L^* has only 1-runs of length one it can be assumed that S^* has this property. The it is easy to verify that $S = S^*1010101$ is a minimal supersequence of L that does not contain substring $T = 11$. □

Theorem 5.7. *The Fixed Non-Substring Minimal Common Non-Subsequence problem can be solved in linear time over any alphabet.*

Proof. Let a string T and a set L of strings over an alphabet Σ be an instance of our problem. Assume $|\Sigma| \geq 2$ (Otherwise the problem is trivial). There exists an $a \in \Sigma$, such that $T \notin a^*$. Let k be the maximum number of a's occurring in a string of L. Then $S = a^{k+1}$ is a minimal non-subsequence of L that does not contain the substring T. □

Definition 5.8. Fixed 2-Non-Substring Minimal Common Non-Subsequence:
Given: A set L of strings and two strings T_1 and T_2 over an alphabet Σ.
Question: Does there exist a minimal supersequence of L that does not contain either of the strings T_1 and T_2 as a substring?

Theorem 5.9. *The Fixed 2-Non-Substring Minimal Common Non-Subsequence problem is NP-complete over an alphabet of size 2 even if the given non-substrings have constant length.*

Proof. We reduce the 3-SAT to our problem. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ be a set of clauses of size three over a set of variables $V = \{v_1, v_2, \ldots, v_n\}$. We define a set L of strings over $\{0, 1\}$ and strings T_1 and T_2 as follows: Let $T_1 = 00000$, $T_2 = 11$, $S_0 = (1^{7n}0)^{7n-1}1^{7n}$. For each variable $v_i \in V$ let $T_i = (10)^{7(i-1)+1}00(10)^{7(n-i)+4}1$, $T_i' = (10)^{7(i-1)+1}0(10)^3 0(10)^{7(n-i)+1}1$, $T_i'' = (10)^{7(i-1)+2}0(10)^2 0(10)^{7(n-i)+1}1$. For each clause C_l let T_{n+l} be a string with $7n$ zeros and $7n - 2$ ones. In T_{n+l} there is a one at both ends and between each two zeros with the following exceptions i) not between the $7(h-1)+1$th and $7(h-1)+2$th zero, if $v_h \in C_l$, ii) not between the $7(h-1)+2$th and $7(h-1)+3$th zero, if $\bar{v}_h \in C_l$. Set $L = \{S_0\} \cup \{T_i, T_i', T_i'' \mid i \in [1 : n]\} \cup \{T_i \mid i \in [n+1 : n+m]\}$. Note that each string in L — with the exception of S_0 — contains exactly $7n$ zeros. Now it can be shown that a $\mathcal{C}$-satisfying truth assignment of V exists when there exists a minimal non-subsequence of L that contains neither of the strings T_1 and T_2 as a substring. □

In all the NP-hardness proofs of this section the string T (respectively the strings T_1 and T_2 in the proof of Theorem 5.9) is not dependent on the instance of the reduced problem. E.g. it is an NP-hard problem to decide for a given set of strings if there exists a minimal

	shortest	longest	fixed substring	fixed non-substring
minimal supersequence	NPc: $\|\Sigma\| = 2$ [5]	MAX SNP: $\|\Sigma\| = 2$ [14]	NPc: $\|\Sigma\| = 2$ Th. 20	NPc: $\|\Sigma\| = 2$ Th. 22
maximal subsequence	MAX SNP: $\|\Sigma\| = 2$ [14]	NPc: $\|\Sigma\| = 2$ [10]	NPc: $\|\Sigma\| = 2$ Th. 20	NPc: $\|\Sigma\| = 2$ Th. 22
maximal non-supersequence	MAX SNP: $\|\Sigma\| = 2$ [14]	NPc: $\|\Sigma\| = 2$ [19]	NPc: $\|\Sigma\| = 2$ Th. 20	NPc: $\|\Sigma\| = 2$ Th. 22
minimal non-subsequence	NPc: $\|\Sigma\| = 2$ [13]	MAX SNP: $\|\Sigma\| = 2$ Th. 17	NPc: $\|\Sigma\| = 2$ Th. 20	$O(n)$, Th. 23 (NPc: 2 fixed non-substr. Th. 25)

Table 3: Complexity of minimal supersequence and maximal subsequence problems: P=polynomial time, NPc=NP-complete, MAX SNP=MAX SNP-hard, n=total length of strings, Σ=alphabet

supersequence that contains the string 11100011000 as a substring. An interesting question is for which strings T the corresponding problems are NP-hard and for which strings the problems become polynomial time solvable. Note, e.g., that the problem to decide whether for a given set L of strings a minimal supersequence exists that contains the substring 0 is polynomial time solvable (There exists such a minimal supersequence iff there exists a string in L that contains a 0).

Table 3 summarizes the complexity results of this section.

6 Open Problems

Some remaining open problems are to characterise the complexities of finding a minimal superstring and of finding a longest (sub-,char-,e-)minimal superstring, given a set of strings. Also it is an interesting question whether the four remaining open problem from the classical minimal/maximal shortest/longest (non)super-/subsequence problems are MAX SNP-hard for strings over alphabet of size 2, i.e. finding a shortest supersequence, a longest subsequence, a longest non-supersequence, and a shortest non-subsequence.

Acknowledgements

We thank the anonymous referees for their detailed and valuable comments. One referee provided ideas that helped to simplify the proofs of Theorems 3.3, 5.4, and 5.6, and to strengthen Theorem 3 so that it holds for an alphabet of size 2.

References

[1] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis, Linear approximation of shortest superstrings, *J. ACM* **41** (1994) 630–647.

[2] M. Farach, Optimal suffix tree construction with large alphabets, *Proceedings of FOCS '97: the 38th Annual IEEE Symposium on Foundations of Computer Science* (IEEE Computer Society, 1997) 137–143.

[3] C.B. Fraser, Subsequences and Supersequences of Strings. *PhD Thesis*, (Dept. of Computing Science, University of Glasgow, 1995).

[4] C.B. Fraser, R.W. Irving, and M. Middendorf, Maximal common subsequences and minimal common supersequences, *Information and Computation* **124** (1996) 145–153.

[5] J. Gallant, D. Maier, and J.A. Storer, On finding minimal length superstrings, *JCCS* **20** (1980) 50–58.

[6] M.R. Garey, D.S. Johnson, *Computers and Intractability*, (W.H. Freeman, San Francisco, 1979).

[7] L.C.K. Hui, Color set size problem with applications to string matching, *Proceedings of CPM '92: the 3rd Annual Symposium on Combinatorial Pattern Matching*, (Springer-Verlag, 1992), LNCS 644, 230–243.

[8] D. Gusfield, *Algorithms on strings, trees and sequences*, (Cambridge University Press, 1997).

[9] T. Jiang and M. Li, On the Approximation of Shortest Common Supersequences and Longest Common Subsequences, *SIAM J. Comput.* **24** (1995) 1122–1139.

[10] D. Maier, The complexity of some problems on subsequences and supersequences, *J. ACM.* **25** (1978) 322–336.

[11] D.F. Manlove, Minimaximal and maximinimal optimisation problems: a partial order-based approach, *PhD thesis*, (Department of Computing Science, University of Glasgow, 1998).

[12] F. Massaglia, P. Poncelet, R. Cischetti, An efficient algorithm for Web usage mining, *Networking and Information Systems Journal*, 2:571-603, 1999.

[13] M. Middendorf, The shortest common nonsubsequence problem is NP-complete, *Theoret. Comput. Sci.*, **108** (1993) 365–369.

[14] M. Middendorf, On finding minimal, maximal, and consistent sequences over a binary alphabet, *Theoret. Comput. Sci.* **145** (1994) 317–327.

[15] M. Middendorf, Plan Merging und verwandte Probleme, *Habilitation thesis*, (Faculty of Economics, University of Karlsruhe, 1998).

[16] C.H. Papadimitrou and M. Yannakakis, Optimization, approximation and complexity classes, *JCCS* **63** (1991) 425–440.

[17] K.-J. Räihä and E. Ukkonen, The shortest common supersequence problem over binary alphabet is NP-complete, *Theoret. Comput. Sci.* **16** (1981) 187–198.

[18] V. G. Timkovsky, Complexity of common subsequence and supersequence problems and related problems, *Cybernetics*, **25:** (1990) 565–580.

[19] L. Zhang, On the approximation of longest common non-supersequences and shortest common non-subsequences, *Theoret. Comp. Sci.* **143** (1995) 353–362.