Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Journal of Algorithms 49 (2003) 192–216

Journal of Algorithms

www.elsevier.com/locate/jalgor

Analogs & duals of the MAST problem for sequences & trees

Michael Fellows,[a,*] Michael Hallett,[b] and Ulrike Stege [c]

[a] *Shool of Electrical Engineering and Computer Science, University of Newcastle, Callaghan 2308 NSW, Australia*

[b] *Shool of Computer Science, McGill University, Montreal, PQ, Canada H3A 2B4*

[c] *Department of Computer Science, University of Victoria, Victoria, BC, Canada V8W 3P6*

Received 21 April 1999

Abstract

Two natural kinds of problems about "structured collections of symbols" can be generally referred to as the LARGEST COMMON SUBOBJECT and the SMALLEST COMMON SUPEROBJECT problems, which we consider here as the dual problems of interest. For the case of rooted binary trees where the symbols occur as leaf-labels and a subobject is defined by label-respecting hereditary topological containment, both of these problems are NP-complete, as are the analogous problems for sequences (the well-known LONGEST COMMON SUBSEQUENCE and SHORTEST COMMON SUPERSEQUENCE problems). When the trees are restricted by allowing each symbol to occur as a leaf-label at most once (which we call a *phylogenetic tree* or *p-tree*), then the LARGEST COMMON SUBTREE problem, better known as the MAXIMUM AGREEMENT SUBTREE (MAST) problem, is solvable in polynomial time. We explore the complexity of the basic subobject and superobject problems for both sequences and binary trees when the inputs are restricted to p-trees and p-sequences (*p-sequences* are sequences where each symbol occurs at most once). We prove that the sequence analog of MAST can be solved in polynomial time. The SHORTEST COMMON SUPERSEQUENCE problem restricted to inputs consisting of a collection of p-sequences (pSCS) is NP-complete; we show NP-completeness of the analogous SMALLEST COMMON SUPERTREE problem restricted to p-trees (pSCT). We also show that both problems are hard for the parameterized complexity classes $W[1]$ where the parameter is the number of input objects. We prove fixed-parameter tractability for pSCS and pSCT when the k input objects are restricted to be complete: every symbol of Σ occurs exactly once in each object and the question is whether there is a common superobject of size bounded by $|\Sigma| + r$ and the parameter

* Corresponding author.
E-mail addresses: mfellows@cs.newcastle.edu.au (M. Fellows), hallett@cs.mcgill.ca (M. Hallett), stege@csr.uvic.ca (U. Stege).

0196-6774/$ – see front matter
doi:10.1016/S0196-6774(03)00081-6

is the pair (k, r). We show that without this restriction, both problems are harder than DIRECTED FEEDBACK VERTEX SET, for which parameterized complexity is famously unresolved.

1. Introduction

This paper is concerned with the complexity of some basic problems regarding structured collections of symbols, namely strings of symbols, and binary trees where the leaves are labeled from a basic set Σ. In both kinds of problems, natural situations arise in which the symbols may or may not be repeated. In the examples below, we discuss two such computational situations. Typically algorithms used for computing phylogenetic relationships are concerned, naturally, with trees where the leaves are labeled from a set Σ of species and each label occurs at most once. However, the study of gene-duplication events leads to models that make use of trees where the species labels may occur more than once.

Example 1 (Modeling gene duplication with repeated leaf-labeled trees). The species tree for a set of taxa reflects the phylogenetic relationships between the given species. When trying to resolve the species tree for a set of n taxa, one typically creates a set of k gene trees, where each gene tree is built over a family of homologous genes of the given taxa. Usually these gene trees do not agree. One reason for the contradictory topologies of the gene trees is due to paralogous duplications of genes followed by subsequent losses. The papers [17,18,21] provide formal models for rectifying the gene trees by the correct species tree with gene-duplication and/or gene-loss events. The rectification can be drawn as a tree using repeated-leaf labels (*rl-tree*). This example is illustrated in Fig. 1.

For problems about sequences, we usually assume that for sequences of interest, symbols will occur many times. But there are some applications where attention may be restricted to sequences x where any symbol occurring in x occurs at most once.

Example 2 (Scheduling manufacturing operations). The SHORTEST COMMON SUPERSEQUENCE problem can be applied to scheduling. Each symbol of the alphabet Σ corresponds to an operation in the sequential process of manufacturing an object. The input to the problem corresponds to the manufacture sequences for a number of different objects that share the same production line. A common supersequence then corresponds to a schedule of operations for the production line that allows all of the different objects to be manufactured. It may reasonably be the case that each object to be manufactured requires any given operation to be applied at most once.

Restricting attention to trees or sequences without repeated symbols seems to have a significant effect on the problem complexity. In order to discuss this issue we introduce the following terminology. A *p-tree* is a rooted tree where the leaves are labeled from an alphabet Σ, and where no symbol in Σ is used more than once as a label. An *rl-tree* is a

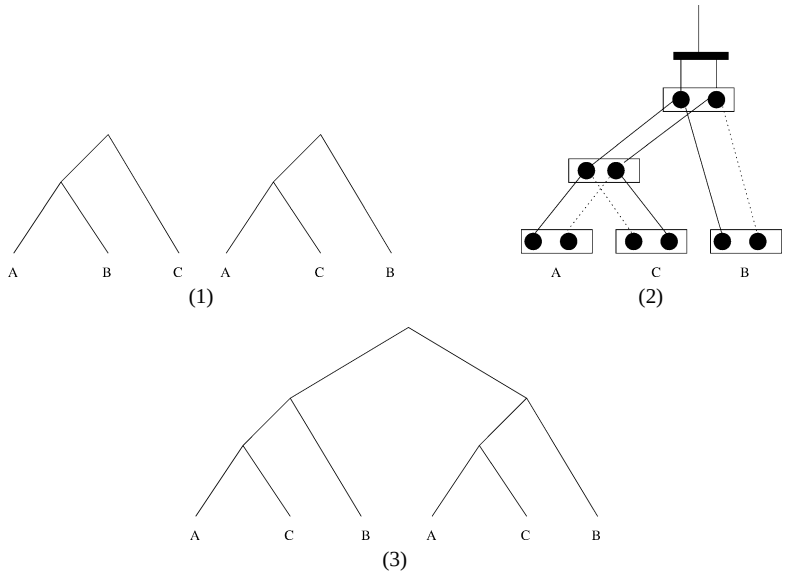

Fig. 1. (1) A gene tree (left) and a species tree. (2) The evolution of the gene family in the duplication/loss model. The black rectangle means a duplication event, the dotted lines show gene losses. A vertex whose kids are both connected with solid lines mean a speciation of the gene simultaneously with the speciation of the taxa. (3) The rl-tree corresponding to the picture in (2).

rooted tree with leaves labeled from Σ, where labels may be repeated. Say that a string of symbols (or sequence) $x \in \Sigma^*$ is a *p-string* (*p-sequence*) if no symbol of Σ occurs more than once in x. We call x a *complete* p-sequence if each symbol of the alphabet occurs exactly once in x. To emphasize the analogy with trees, say that x is an *rl-string* if it is a string in the usual sense, where symbols of Σ may be repeated. If x and y are strings, then we write $x \subseteq y$ to denote that x is a subsequence of y. If X and Y are rl-trees, then $X \subseteq Y$ denotes that X is contained in Y by topological containment that respects ancestry with label isomorphism at the leaves. For a given tree T, the *size* $|T|$ of T is defined by the number of leaves in T.

The following quartet of fundamental computational problems are natural and important for various applications.

Two problems about trees

LARGEST COMMON SUBTREE (LCT)
Input: rl-trees $T_1, \ldots, T_k$ and a positive integer m.
Question: Is there an rl-tree T of size $|T| \geqslant m$ leaves, with $T \subseteq T_i$ for $i = 1, \ldots, k$?

SMALLEST COMMON SUPERTREE (SCT)
Input: rl-trees $T_1, \ldots, T_k$ and a positive integer m.
Question: Is there an rl-tree T of size $|T| \leqslant m$ leaves, with $T_i \subseteq T$ for $i = 1, \ldots, k$?

Two problems about sequences

LONGEST COMMON SUBSEQUENCE (LCS)
Input: rl-sequences $x_1, \ldots, x_k$ and a positive integer m.
Question: Is there an rl-sequence x, with $|x| \geqslant m$ and $x \subseteq x_i$ for $i = 1, \ldots, k$?

SHORTEST COMMON SUPERSEQUENCE (SCS)
Input: rl-sequences $x_1, \ldots, x_k$ and a positive integer m.
Question: Is there an rl-sequence x, with $|x| \leqslant m$ and $x_i \subseteq x$ for $i = 1, \ldots, k$?

All four of these problems are NP-complete (see [19] for the sequence problems; we prove NP-completeness for the tree problems in Section 3). Both LCS and SCS can be solved in $O(n^k)$ time [19,26]. LCT and SCT, the tree analogs of LCS and SCS, have similar polynomial complexity for fixed k (Section 3). The question we explore is

> What happens to the complexity of these basic problems when the inputs are restricted to be p-trees and p-sequences?

The LARGEST COMMON SUBTREE problem restricted to p-trees is better known as the MAXIMUM AGREEMENT SUBTREE (MAST) problem. This is an important problem for which useful polynomial-time algorithms (for trees of bounded degree) have recently been developed. In the study of evolution the situation frequently arises that one has k sets of gene sequence data for n species. A typical approach would be to compute *gene trees* for each of the data sets. For various reasons, these trees will frequently not be in agreement. One may then use an algorithm for MAST to compute a maximum subtree (*species tree*) on which the gene trees agree. Farach et al. have described an algorithm for MAST for k rooted p-trees for n species of maximum degree d that runs in time $O(kn^3 + n^d)$ [14]. A different (and simpler) algorithm with the same time complexity was developed by Bryant [4]. Przytycka has described a modification of the algorithm of [14] that runs in time $O(kn^3 + k^d)$ [22]. The described results on MAST answer our question for one of the four basic problems, but what of the other three?

Main results

- We describe an $O(k|\Sigma|(k + \log|\Sigma|))$-time algorithm to solve the LONGEST COMMON SUBSEQUENCE problem for inputs restricted to p-sequences (cf. Section 3).
- We prove that SHORTEST COMMON SUPERSEQUENCE and SMALLEST COMMON SUPERTREE for inputs restricted to p-sequences and p-trees remain NP-complete (cf. Section 3), and are hard for $W[1]$ when the parameter is the number of input objects (cf. Section 4).
- We prove fixed-parameter tractability for SHORTEST COMMON SUPERSEQUENCE for p- and rl-sequences when the question is whether there is a common supersequence of length at most $|\Sigma| + r$ parameterized by the number of sequences and r. We present an algorithm with running time $O(k^r \cdot n)$ (cf. Section 5).
- We prove fixed-parameter tractability for SHORTEST COMMON SUPERSEQUENCE for complete p-sequences and parameter r when the question is whether there is a common

supersequence of length $|\Sigma| + r$. In fact, we prove tractability for a more general, weighted alphabet form of the problem (cf. Section 5).

- We prove fixed-parameter tractability for SHORTEST COMMON SUPERTREE for k complete p-trees parameterized by the pair (k, r), when the question is whether there is a common supertree of size $|\Sigma| + r$ (cf. Section 5).

For those who are not familiar with parameterized complexity we give a small discussion in the following section.

2. The parameterized-complexity framework

Part of our analysis of the complexity of the problems that we look at is in the framework of parameterized complexity introduced in [8–11] that has become a routine tool to some extent, but not to the point where we feel it can be assumed to be familiar, and so we offer this brief summary. The parametric framework has previously been applied to sequence problems in [2,3].

The goal in parametric complexity analysis is to identify small but still useful ranges of a *parameter* k for a hard problem, and see if for instances of size n the problem can be solved in time $f(k)n^{\alpha}$ for some constant α independent of the parameter. (Typically, when this is possible, α is small, just as with polynomial time complexity.) This good behavior is termed *fixed-parameter tractability*. For surveys about fixed-parameter tractability see [10,12].

A canonical example that exhibits the issue is the difference between the best known algorithms for VERTEX COVER and DOMINATING SET, taking in both cases the parameter to be the number of vertices in the set. The best known algorithm for VERTEX COVER, after several rounds of improvement in the parameter function, runs in time $O(kn + \max\{(1.25542)^k k^2, (1.286)^k k\})$ [7,23,24]. Note that the exponential dependence on the parameter k in the last expression is *additive*, so that VERTEX COVER is well-solved for input of any size so long as k is no more than around 60. Both problems are NP-complete, of course, and so the P-time *versus* NP-complete framework is unable to detect the significant qualitative difference between these two problems displayed in Table 1.

For parameterized complexity, the analog of NP-hardness is hardness for $W[1]$. The analogy is very strong, since the k-STEP HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES is complete for $W[1]$ [6], a result which is essentially a miniaturization of Cook's Theorem. DOMINATING SET is hard for $W[1]$ and is therefore unlikely to be

Table 1
The ratio $n^{k+1}/(2^k n)$ for various values of n and k

	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2,500	5,625
$k = 3$	15,625	125,000	421,875
$k = 5$	390,625	6,250,000	31,640,625
$k = 10$	1.9×10^{12}	9.8×10^{14}	3.7×10^{16}
$k = 20$	1.8×10^{26}	9.5×10^{31}	2.1×10^{35}

fixed-parameter tractable. Indeed, one senses a "wall of brute force" (try all k-subsets) inherent complexity in this problem, much as one does for the k-STEP HALTING PROBLEM, where a significant improvement on the obvious $O(n^k)$ algorithm seems fundamentally unlikely, and much as one senses a wall of brute force necessity in the complexity of NP-complete problems such as SATISFIABILITY, where we would be similarly surprised if an algorithm running in time $O(2^{o(n)})$, for n variables, were possible.

3. The classical complexity of the four basic problems

In this section we study the classical complexity of the four basic problems SCS, LCS, SCT and LCT in the unrestricted versions and restricted to p-objects (cf. Table 2).

3.1. The classical complexity of SCS, LCS, SCT, and LCT

The complexity situation for the tree analogs SCT and LCT of the sequence problems SCS and LCS turns out to be almost identical to the situation for these well-studied sequence problems. Both SCT and LCT are NP-complete, but can be solved for k trees by k-dimensional dynamic programming. We prove the NP-completeness of LCT via a reduction from LCS. We receive the NP-completeness of SCT with a reduction from SCS, for which we give an easier NP-completeness proof than [19] via the NP-complete problem DIRECTED FEEDBACK VERTEX SET (DFVS) [16]. DIRECTED FEEDBACK VERTEX SET is stated as follows.

DIRECTED FEEDBACK VERTEX SET (DFVS)
Input: A directed graph $D = (V, A)$ and a positive integer l.
Question: Is there a set $V' \subseteq V$ with $|V'| \leqslant l$ such that $D - V'$ is acyclic?

Theorem 1. *SCS is NP-complete.*

Table 2
The classical complexity of the 4 basic problems

	Unrestricted input	Input are p-objects
LCT	NP-complete	polynomial
	$O(n^{k+1})$, $n = \max \lvert T_i \rvert$	
	Theorems 3 and 5	[4,14,22]
SCT	NP-complete	NP-complete
	$O(n^{k+1})$, $n = \max \lvert T_i \rvert$	
	Theorems 2 and 4	Theorem 7
LCS	NP-complete	polynomial
	$O(n^k)$, $n = \lvert \Sigma \rvert$	
	[19]	Theorem 6
SCS	NP-complete	NP-complete
	$O(n^k)$, $n = \lvert \Sigma \rvert$	
	[19]	[20,25]

Proof. Let $D = (V, A)$, $|V| = n$, be a digraph for which we wish to determine if it has a directed feedback vertex set of size l. The instance of SCS to which we transform this is described as follows. The alphabet is V. Each arc uv of D becomes the length 2 sequence uv. The set of sequences S thus has the same size as the set A of arcs of D. The parameter m of the image instance is equal to l.

Let x be a solution for the sequence problem, and let V' be the set of vertices (which double as the symbols of the alphabet) that occur more than once in x. Clearly $|V'| \leqslant l$. We argue that V' covers all the directed cycles in D. If not, then there is a directed cycle C in D involving vertices that occur exactly once in x. Let a be the first vertex of C that occurs in x. But then there is some vertex b of C such that ba is an arc of C and therefore ba is a sequence in S. Since b occurs after a in x this contradicts that x is a solution to the SCS problem.

Conversely, let V' be a l-element directed feedback vertex set for D. A common supersequence x for the sequences in S can be written $x = x_1x_2x_3$ where $x_1 = x_3$ is any permutation of V' and x_2 is a topological sort of $V - V'$. If uv is an arc of D where both u and v belong to V', then clearly uv is a subsequence of x, either by finding u in x_1 and v in x_3 or vice versa. If both u and v belong to $V - V'$ then uv is a subsequence of x_2. The two remaining cases (where exactly one of u and v belongs to V') are equally easy to check. □

Theorem 2. *SCT is NP-complete.*

Proof. We reduce from SCS. An instance of SCS consists of a set $\mathcal{X}$ of sequences over an alphabet Σ, $|\Sigma| = n$, and a positive integer m. We define a complete binary caterpillar tree F of size $n + m + 1$ which is formed over the leafset $L_F = \{f[1], f[2], \ldots, f[n + m + 1]\}$, $L_F \cap \Sigma = \emptyset$, such that $f[2]$ is sibling of $f[1]$, $f[3]$ is sibling of the ancestor of $f[1]$ and $f[2]$, and so on. Furthermore we introduce a new symbol $\$ \notin \Sigma \cup L_F$. A sequence $x_i \in \mathcal{X}$ of length t_i, where x_i consists of the sequence of symbols of $x_i = x_i[1]\cdots x_i[t_i]$ is transformed to a tree T_i represented by the parenthesized expression $(((\cdots((F, x_i[1]), x_i[2])\cdots)x_i[t_i]), \$)$. The alphabet of leaf labels of the trees is thus $\Sigma' = \Sigma \cup L_F \cup \{\$\}$. Let $m' = n + 2(m + 1)$.

Let $T = (V, E, L)$, $|T| \leqslant m'$, be a solution for the tree problem. Since T is a supertree of all the T_i, $1 \leqslant i \leqslant k$, we can assume that the vertex in T being the least common ancestor of all the elements in L_F induces a subtree of T with a leafset equals L_F. Furthermore, we can reorganize T as a caterpillar tree. This follows straightforward from the fact, that all the T_i are caterpillar trees. Thus we can assume T having the form $(((\cdots((F, x[1]), x[2])\cdots)x[m' - 1 - (n + m + 1)]), \$)$ and obviously $x = x[1]x[2]\cdots x[m' - 1 - (n + m + 1)]$ is a common supersequence of the x_i, $1 \leqslant i \leqslant k$, of length $|x| = m' - 1 - (n + m + 1) = n + 2(m + 1) - 1 - n - m - 1 = m$.

The other direction is straightforward and left to the reader. □

Theorem 3. *LCT is NP-complete.*

Proof. We do the following straightforward construction from LCS. Let $x_1 \ldots x_k, m$ be an instance of LCS. Each x_i can be written as $x_i = x_i[1]\ldots x_i[n_i]$, $1 \leqslant i \leqslant k$. For

each sequence x_i $(1 \leqslant i \leqslant k)$ we construct a caterpillar tree T_i with leafset $L_i = \{x_i[1], \ldots, x_i[n_i]\}$ such that $x_i[2]$ is sibling of $x_i[1]$ and $x_i[3]$ is sibling of the least common ancestor of $x_i[1]$ and $x_i[2]$ and so on.

We show the largest common subtree T of $T_1, \ldots, T_k$ has size $|T| \leqslant m$ if and only if the longest common subsequence x of $x_1, \ldots, x_k$ has size $|x| \leqslant m$. Let the sequence $x = x[1]x[2]\ldots x[m]$ be a solution of the LCS instance. We construct the caterpillar tree T over the leafset $L = \{x[1], x[2], \ldots, x[m]\}$ such that $x[2]$ is sibling of $x[1]$ and $x[3]$ is sibling of the least common ancestor of $x[1]$ and $x[2]$ and so on. Because of the caterpillar topology it is easy to see that T is subtree of every T_i $(1 \leqslant i \leqslant k)$. It remains to show that there is no tree $T' \neq T$ which is a common subtree of each T_i $(1 \leqslant i \leqslant k)$ and $|T'| > |T|$. For the sake of contradiction we assume there is such a T' over leafset $L' = \{x'[1], \ldots, x'[m']\}$ with $|T'| > |T|$ and therefore $m' > m$. Because T' is a subtree of the T_i we know T' has the topology of a caterpillar tree. We assume T' is built in the usual way, i.e., $x'[2]$ is sibling of $x'[1]$ and $x'[3]$ is sibling of the least common ancestor of $x'[1]$ and $x'[2]$ and so on. But then $x' = x'[1]x'[2]\ldots x'[m']$ is a subsequence of x_i $(1 \leqslant i \leqslant k)$. This contradicts since $|x'| > |x|$.

For the other direction let T be the largest common subtree over the leafset $L = \{x[1], x[2], \ldots, x[m]\}$. Here the sequence $x = x[1]x[2]\ldots x[m]$ is a solution for the sequence problem. The argument is similar to the one above and left to the reader. □

The following two theorems are consequences from a straightforward k-dimensional dynamic programming.

Theorem 4. *For each fixed k, SCT can be solved in time $O(n^{k+1})$ where $n = \max |T_i|$.*

Theorem 5. *For each fixed k, LCT can be solved in time $O(n^{k+1})$ where $n = \max |T_i|$.*

3.2. The classical complexity of SCS, LCS, SCT, and LCT restricted to p-objects

We consider the problems restricted to p-objects.

LONGEST COMMON SUBSEQUENCE FOR P-SEQUENCES (PLCS)
Input: p-sequences $x_1, \ldots, x_k$ over an alphabet Σ, and $m \in Z^+$.
Question: Is there a p-sequence x, $|x| \geqslant m$, such that $x \subseteq x_i$ for $i = 1, \ldots, k$?

LARGEST COMMON SUBTREE FOR P-TREES (PLCT/MAST)
Input: Binary p-trees $T_1, \ldots, T_k$ and a positive integer m.
Question: Is there an rl-tree T of size $|T| \geqslant m$ leaves, with $T \subseteq T_i$ for $i = 1, \ldots, k$?

P-SEQUENCE SHORTEST COMMON SUPERSEQUENCE (PSCS)
Input: p-sequences $x_1, \ldots, x_k$ and a positive integer M.
Question: Is there an rl-sequence x, with $|x| \leqslant M$ and $x_i \subseteq x$ for $i = 1, \ldots, k$?

SHORTEST COMMON SUPERTREE FOR P-TREES (PSCT)
Input: Binary p-trees $T_1, \ldots, T_k$ and a positive integer m.
Question: Is there an rl-tree T, with $|T| \leqslant m$ and $T_i \subseteq T$ for $i = 1, \ldots, k$?

While not difficult, the positive result of the PLCS, the analog of MAST for p-sequences, shows an interesting parallel between the complexity of sequence problems and the complexity of leaf-labeled tree problems that we will see holds up also for the superobject problems.

Theorem 6. *PLCS can be solved in time $O(kn(k+\log n))$ for an alphabet of size n.*

Proof. Let $\Sigma = \{1, 2, \ldots, n\}$. W.l.o.g. let $|x_i| = |x_j| = n$ for all $1 \leqslant i, j \leqslant k$, $i \neq j$, and let x_1 be the p-sequence $1\,2\,3 \ldots n$. Let $x_i[j]$ denote the jth symbol in sequence x_i. Furthermore for $a \in \Sigma$ let $x_i^{-1}(a) = j$ if $x_i[j] = a$. We construct a directed graph $G = (\Sigma, E)$ with $E = \bigcup_{i=1}^{k} E_i$. Let $E_1 = \{(a, a+1) \mid a \in \Sigma \setminus \{n\}\}$. For elements $a, b \in \Sigma$ we define $(a, b) \in E_i$ if and only if (1) $a < b$ and (2) b is the smallest symbol in x_i after the occurrence of a. We can compute G in time $O(nk \log n)$.

From G we construct a directed graph $G' = (\Sigma, E')$ such that $(a, b) \in E'$ if and only if (1) $(a, b) \in E$ and (2) $x_i^{-1}(b) - x_i^{-1}(a) > 0$ for all $i = 1, \ldots, k$. Computing G' is possible in time $O(k^2 n)$. Thus, G' is a partial ordered graph having at most n arcs. Topological sorting of the arcs in G' computes the longest common subsequence. Since topological sorting can be done in $O(n)$ time [1], the overall running time is $O(kn(k+\log n))$. □

The NP-completeness of SCS restricted to p-sequences is shown by Timkovsky [25] and Middendorf [20] even restricted to sequences of length 2. For the NP-completeness of PSCT note that the reduction in Theorem 2 holds even restricted to binary p-trees.

Theorem 7. *PSCT is NP-complete.*

4. Intractable parameterizations

We show $W[1]$-hardness for SCS and SCT when restricted to p-objects and parameterized by the number of input objects.

P-SEQUENCE SHORTEST COMMON SUPERSEQUENCE (k-PSCS)
Input: p-sequences $x_1, \ldots, x_k$ and a positive integer M.
Parameter: k.
Question: Is there an rl-sequence x, with $|x| \leqslant M$ and $x_i \subseteq x$ for $i = 1, \ldots, k$?

SMALLEST COMMON SUPERTREE FOR P-SEQUENCES (k-PSCT)
Input: binary p-trees $T_1, \ldots, T_k$ and a positive integer m.
Parameter: k.
Question: Is there an rl-tree T, with $|T| \leqslant m$ and $T_i \subseteq T$ for $i = 1, \ldots, k$?

The hardness of k-PSCS is shown by a reduction from the problem k-CLIQUE.

k-CLIQUE
Input: A graph $G = (V, E)$, a positive integer k.
Parameter: k.

Question: Is there a set of k vertices $V' \subseteq V$ that forms a complete subgraph of G (that is, a clique of size k)?

This parameterized version of CLIQUE [16] is known to be $W[1]$-complete [9]. The hardness for k-PSCT is a straightforward reduction from PSCS.

Theorem 8. *PSCS is hard for $W[1]$ when parameterized by the number of sequences k.*

Proof. We prove the theorem by a reduction from k-CLIQUE. Let $V = \{1, 2, \ldots, n\}$, $|E| = m$, $\alpha = \binom{k}{2}$. W.l.o.g. let $m \geqslant 3$. Let the bijection σ defined as $\sigma : E \to \{1, \ldots, m\}$ with $\sigma((i, j)) < \sigma((i', j'))$ if and only if either $i < i'$ or ($i = i'$ and $j < j'$). Furthermore if $\sigma((i, j)) = e$ then $\sigma_1^{-1}(e) = i$, $\sigma_2^{-1}(e) = j$. We define $f_1(i) = i + 1 + (i - 1)k - \binom{i}{2}$, $f_2(i, j) = j - k + i \cdot k - \binom{i+1}{2}$, and we write $\prod_{i=1}^{k} x_i$ to denote the sequence $x_1 x_2 \cdots x_k$.

The idea for the reduction from k-CLIQUE to SCS is sketched as follows. We first ignore the fact that we need to transform the graph into p-sequences. Then we modify this construction to achieve a reduction to p-sequences. Three basic gadgets supply the building blocks for our reduction: a "vacuum cleaner" building the frame sequence, an exhibition mechanism for the edge selection for possible clique edges (α sequences, each encoding the graph represented by the edges), and another gadget for the checking the validity of selected edges. The vacuum cleaner Vac,

$$\text{Vac: } \mathcal{X}\mathcal{Y}\,\text{CliqueSelection}\,\mathcal{Y}\mathcal{X} \quad \left(\mathcal{X}\text{: } \mathcal{X}^1, \ldots, \mathcal{X}^m, \mathcal{Y}\text{: } \mathcal{Y}^1, \ldots, \mathcal{Y}^n\right),$$

forces exactly one edge of each edge-selection sequence

$$\text{CliqueEdge}(i)\text{: } \mathcal{X}^1 \#_{i-1} \sigma_1^{-1}(1) \mathcal{N}(1) \sigma_2^{-1}(1) \#_i \mathcal{X}^2 \#_{i-1} \sigma_1^{-1}(2) \mathcal{N}(1) \sigma_2^{-1}(2) \#_i \cdots \mathcal{X}^{m-1} \#_{i-1} \sigma_1^{-1}(m) \mathcal{N}(1) \sigma_2^{-1}(m) \#_i \mathcal{X}^m$$

$(1 \leqslant i \leqslant \alpha)$, to land in the CliqueSelection-zone of each shortest common supersequence. The other edges are "absorbed" by $\mathcal{X}$. The CliqueSelection-zone is "preprocessed" by having the following form.

$$\text{CliqueSelection: } \#_0 \mathcal{N}(1) \#_1 \mathcal{N}(2) \#_2 \cdots \mathcal{N}(m) \#_m.$$

Thus, each selected edge fits in it's corresponding range. The first edge is selected from CliqueEdge(1), the second from CliqueEdge(2) and so on.

The checking mechanism works tuplewise. Consider a clique of size k in G, $\{u_1, u_2, \ldots, u_k\} \subseteq V$, $u_1 < u_2 < \cdots < u_k$. The edges of the clique can be ordered in the following way (cf. the definition of σ):

$$(u_1, u_2), (u_1, u_3), \ldots, (u_1, u_k), (u_2, u_3), (u_2, u_4), \ldots, (u_2, u_k), \ldots, (u_{k-1}, u_k).$$

Each of the k clique-vertices occurs in exactly $(k - 1)$ of the α clique-edges. If the clique is represented as the sequences before we get

$$\#_0 u_1 \mathcal{N}(1) u_2 \#_1 u_1 \mathcal{N}(2) u_3 \#_2 \cdots \#_{k-1} u_{k-1} \mathcal{N}(k) \#_k,$$

and for a single clique-vertex l we receive the sequence

$$\text{CliqueVertex}(l)\colon \prod_{i=1}^{l-1}\left[\#_{f_2(i,l)-1}\mathcal{N}\big(f_2(i,l)\big)u_l\#_{f_2(i,l)}\right] \times \prod_{i=l}^{k-1}\left[\#_{f_1(l)+i-l-1}u_l\mathcal{N}\big(f_1(l)+i-l\big)\#_{f_1(l)+i-l}\right].$$

For checking, whether one vertex in V fulfills the property to be the lth clique vertex we get

$$\text{Check}(l)\colon \prod_{j=n}^{1}\left[\text{CliqueVertex}(l)\right].$$

Thus, a shortest common supersequence has minimal length, if the selected edges form a clique (the nonclique-vertices for each sequence are accumulated by $\mathcal{Y}$). The essential part of the shortest supersequence is the part started by the first symbol of the CliqueSelection-substring of Vac and ended by the last CliqueSelection-symbol. This part reflects the differences in the length of the possible shortest common supersequences depending if the graph has a clique of size k or not. This idea carries out using p-sequences in the following way: We split the sequence Vac in two sequences, VacI and VacII, before the first time a symbol is repeated. We add (in the end of VacI and in the beginning of VacII) a GlueingZone, big enough that there is no influence on the "shape" of Vac.

For avoiding repeated symbols in the CliqueEdge(i)-sequences we allow each vertex j to be represented by m different symbols and use the cth one if occurring in edge c. Similarly, we treat the # and $\mathcal{N}$ symbols. To make the checking possible, we have to add for both vertices j and j' of each edge all the not used representations of j and j' in the CliqueSelection-zone. This allows us to use in the checking sequences each different representation of a vertex j depending on the location in the corresponding $(k-1)$-tuple.

Following the idea described above, this leads us to the following gadget consisting of the set of p-sequences Seq = {VacI, VacII, CE(i), $\mathcal{F}(i)$, Check(l)}, $1 \leqslant i \leqslant \alpha,\ 1 \leqslant l \leqslant k$ (as an example see Fig. 2). The alphabet is

$$\begin{aligned}\Sigma = {} & \big\{\mathcal{X}(i,c,f) \,\big|\, 1 \leqslant i \leqslant \alpha,\ 0 \leqslant c \leqslant m,\ 1 \leqslant f \leqslant 4n+5\big\} \\ & \cup \big\{\mathcal{Y}(l,j,f) \,\big|\, 1 \leqslant l \leqslant k,\ 0 \leqslant j \leqslant n,\ 1 \leqslant f \leqslant 4(2k+km-m)\big\} \\ & \cup \big\{\mathcal{Z}(i,c,f) \,\big|\, 1 \leqslant i \leqslant \alpha,\ 0 \leqslant c \leqslant m,\ 1 \leqslant f \leqslant 4(n+m+1)\big\} \\ & \cup \{\mathbf{S},\mathbf{E}\} \\ & \cup \big\{\#(i,c,j) \,\big|\, 0 \leqslant i \leqslant \alpha,\ 1 \leqslant c \leqslant m,\ 1 \leqslant j \leqslant n+2\big\} \\ & \cup \big\{\mathcal{N}(i,c,j,d) \,\big|\, 1 \leqslant i \leqslant \alpha,\ 1 \leqslant c \leqslant m,\ 1 \leqslant j \leqslant n,\ 1 \leqslant 2 \leqslant 2\big\} \\ & \cup \big\{G(f) \,\big|\, 1 \leqslant f \leqslant 2k^2\big(m(2n+4+m)+4(n+2)\big) - 2km(n+m+1) \\ & \quad - 2k(n+1)\big\} \\ & \cup \big\{a(j,c_1,c_2,d) \,\big|\, 1 \leqslant j \leqslant n,\ 1 \leqslant c_1,c_2 \leqslant m,\ 1 \leqslant d \leqslant 2\big\}.\end{aligned}$$

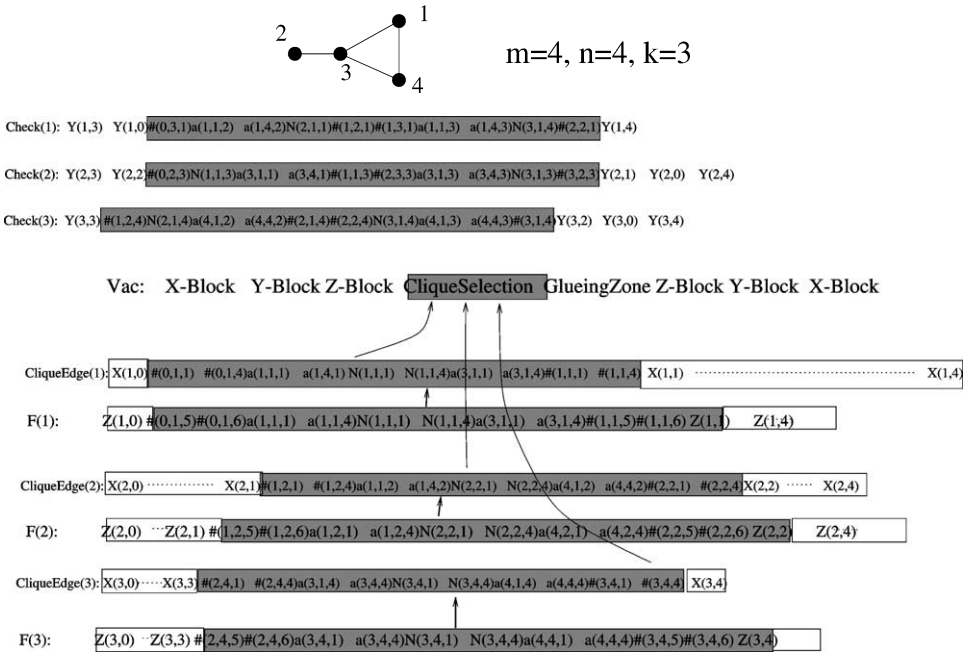

Fig. 2. Example for the reduction of Theorem 8.

Thus, $|\mathrm{Seq}| = 2(\alpha+1)+k$ and

$$|\Sigma| = 6+\alpha\left(23nm+4m^2\right)-2k(19m+9+8n)+2k^2(53+36n+31m)+2nm^2 + 4(n+m+nmk).$$

The sequences are built as follows. We start with vacuum cleaner Vac, which includes three "absorbers" $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$.

VacI: $\mathcal{X}\ \mathcal{Y}\ \mathcal{Z}$ CliqueSelection GlueingZone;

VacII: GlueingZone $\mathcal{Z}\ \mathcal{Y}\ \mathcal{X}$;

$$\mathcal{X}\colon \prod_{i=1}^{\alpha}\prod_{c=0}^{m}\mathcal{X}(i,c);$$

$$\mathcal{X}(i,c)\colon \prod_{f=1}^{4n+5}\mathcal{X}(i,c,f);$$

$$\mathcal{Y}\colon \prod_{l=1}^{k}\left[\prod_{j=n-1}^{0}\mathcal{Y}(l,j)\mathcal{Y}(l,n)\right];$$

$$\mathcal{Y}(l,j)\colon \prod_{f=1}^{4(2k+km-m)}\mathcal{Y}(l,j,f);$$

$$\mathcal{Z}\colon \prod_{i=1}^{\alpha}\prod_{c=0}^{m}\mathcal{Z}(i,c);$$

$$\mathcal{Z}(i,c)\colon \prod_{f=1}^{4(n+m+1)}\mathcal{Z}(i,c,f);$$

$$\text{CliqueSelection}\colon \mathbf{S}\prod_{i=1}^{\alpha}\text{SelectBox}(i)\#(\alpha)\mathbf{E};$$

$$\mathbf{S}\colon \prod_{l=1}^{k}\mathcal{H}(1,l);$$

$$\text{SelectBox}(i)\colon \#(i-1)\mathcal{N}(i);$$

$$\mathbf{E}\colon \prod_{l=1}^{k}\mathcal{H}(2,l);$$

$$\mathcal{H}(i,l)\colon \prod_{f=1}^{2n+m+1}\mathcal{H}(i,l,f);$$

$$\#(i)\colon \prod_{c=1}^{m}\prod_{j=1}^{n+2}\#(i,c,j);$$

$$\mathcal{N}(i)\colon \prod_{c=1}^{m}\prod_{j=1}^{n}\mathcal{N}(i,c,j);$$

$$\mathcal{N}(i,c,j)\colon \prod_{f=1}^{2m}\mathcal{N}(i,c,j,f);$$

$$\text{GlueingZone}\colon \prod_{f=1}^{\delta}G(f),$$

$$\delta = 2k\big[k\big(m(2n+4+m)+4(n+2)\big)-m(n+m+1)-n-1\big].$$

The ith clique-edge is selected from CliqueEdge(i) with absorber $\mathcal{X}$.

$$\text{CliqueEdge}(i)\colon \prod_{c=1}^{m}\Bigg[\mathcal{X}(i,c-1)\prod_{j=1}^{n}\#(i-1,c,j)\prod_{d=1}^{m}a\big(\sigma_1^{-1}(c),d,c\big)\prod_{j=1}^{n}\mathcal{N}(i,c,j) \times\prod_{d=1}^{m}a\big(\sigma_2^{-1}(c),d,c\big)\prod_{j=1}^{n}\#(i,c,j)\Bigg]\mathcal{X}(i,m)$$

$(1\leqslant i\leqslant\alpha)$. With absorber $\mathcal{Z}$ we select from each $\mathcal{F}(i)$ an edge to "fill up" the necessary vertex-symbols for making the checking possible.

$$\mathcal{F}(i)\colon \prod_{c=1}^{m}\Bigg[\mathcal{Z}(i,c-1)\#(i-1,c,n+1)\#(i-1,c,n+2)\prod_{d=1}^{m}a\big(\sigma_1^{-1}(c),c,d\big)\times\prod_{j=1}^{n}\mathcal{N}(i,c,j)\prod_{d=1}^{m}a\big(\sigma_2^{-1}(c),c,d\big)\#(i,c,n+1)\#(i,c,n+2)\Bigg]\mathcal{Z}(i,m)$$

$(1\leqslant i\leqslant\alpha)$. The checking forces with absorber $\mathcal{Y}$ one clique-tuple to be in the Clique-Selection-zone.

$$\text{Check}(l)\colon \prod_{j=n}^{1}\Bigg[\mathcal{Y}(l,j-1)\prod_{i=1}^{l-1}\Bigg[\#\big(f_2(i,l)-1,2,j\big)\mathcal{N}\big(f_2(i,l),1,j\big)\times\prod_{c=1}^{m}a\big(j,c,f_2(i,l)\big)\#\big(f_2(i,l),1,j\big)\Bigg]\times\prod_{i=l}^{k-1}\Bigg[\#\big(f_1(l)+i-l-1,3,j\big)\prod_{c=1}^{m}a\big(j,c,f_1(l)+i-l\big)\times\mathcal{N}\big(f_1(l)+i-l,1,j\big)\#\big(f_1(l)+i-l,2,j\big)\Bigg]\Bigg]\mathcal{Y}(l,n)$$

$(1\leqslant l\leqslant k)$. Note that all the sequences given above are p-sequences. We show G has a k-clique if and only if the sequences given above have a common supersequence of size $|\mathcal{S}|\leqslant M$, $M=2(10+8n+m)+mn+\alpha(53nm+24m^2-4m)+2k(4m+7-3n+nm)+2k^2(8+14n+11m)$. Let $V'\subseteq V$ be a clique of size k in G. We define the bijection $\rho\colon V'\to\{1,\ldots,k\}$, $\rho(x)<\rho(y)$ if and only if $x<y$. Let $\rho^{-1}=u$, and let $u_i=u(i)$, $i=1\leqslant k$. Then $V'=\{u_1,\ldots,u_k\}$. Let the bijection τ be defined as $\tau\colon\{(u_i,u_j)\mid i\neq j, i<j\}\to\{1,\ldots,\alpha\}$, $\tau((u_i,u_j))<\tau((u_{i'},u_{j'}))$ if and only if $i<i'$ or ($i=i'$ and $j<j'$). We show there exist a supersequence $\mathcal{S}$ of size M. $\mathcal{S}$ is built as follows:

$$\begin{aligned}\mathcal{S}\colon\ &\prod_{i=1}^{\alpha}\Bigg[\prod_{c=1}^{m}\big[\mathcal{X}(i,c-1)\diamondsuit_1(i,c)\big]\mathcal{X}(i,m)\Bigg]\cdot\prod_{l=1}^{k}\Bigg[\prod_{j=n}^{1}\big[\mathcal{Y}(l,j-1)\clubsuit_1(l,j)\big]\mathcal{Y}(l,n)\Bigg]\\&\times\prod_{i=1}^{\alpha}\Bigg[\prod_{c=1}^{m}\big[\mathcal{Z}(i,c-1)\spadesuit_1(i,c)\big]\mathcal{Z}(i,m)\Bigg]\\&\cdot\mathbf{S}\prod_{i=1}^{\alpha}\Bigg[\#(i-1)\prod_{c=1}^{m}\prod_{d=1}^{m}a\big(\tau_1^{-1}(i),c,d\big)\mathcal{N}(i)\prod_{c=1}^{m}\prod_{d=1}^{m}a\big(\tau_2^{-1}(i),c,d\big)\Bigg]\\&\times\#(\alpha)\mathbf{E}\,\text{GlueingZone}\\&\times\prod_{i=1}^{\alpha}\Bigg[\prod_{c=1}^{m}\big[\mathcal{Z}(i,c-1)\spadesuit_2(i,c)\big]\mathcal{Z}(i,m)\Bigg]\\&\cdot\prod_{l=1}^{k}\Bigg[\prod_{j=n}^{1}\big[\mathcal{Y}(l,j-1)\clubsuit_2(l,j)\big]\mathcal{Y}(l,n)\Bigg]\\&\times\prod_{i=1}^{\alpha}\Bigg[\prod_{c=1}^{m}\big[\mathcal{X}(i,c-1)\diamondsuit_2(i,c)\big]\mathcal{X}(i,m)\Bigg]\end{aligned}$$

with

$$\diamondsuit_1(i,c)=\begin{cases}\text{Edge}(i,c) & \text{if } \sigma\left(\tau^{-1}(i)\right)>c,\\ \epsilon & \text{otherwise,}\end{cases}$$

$$\diamondsuit_2(i,c)=\begin{cases}\text{Edge}(i,c) & \text{if } \sigma\left(\tau^{-1}(i)\right)<c,\\ \epsilon & \text{otherwise,}\end{cases}$$

$$\clubsuit_1(l,j)=\begin{cases}\text{Vertex}(l,j) & \text{if } j<u_l,\\ \epsilon & \text{otherwise,}\end{cases}$$

$$\clubsuit_2(l,j)=\begin{cases}\text{Vertex}(l,j) & \text{if } j>u_l,\\ \epsilon & \text{otherwise,}\end{cases}$$

$$\spadesuit_1(i,c)=\begin{cases}\text{Filler}(i,c) & \text{if } \sigma\left(\tau^{-1}(i)\right)>c,\\ \epsilon & \text{otherwise,}\end{cases}$$

$$\spadesuit_2(i,c)=\begin{cases}\text{Filler}(i,c) & \text{if } \sigma\left(\tau^{-1}(i)\right)<c,\\ \epsilon & \text{otherwise;}\end{cases}$$

$$\text{Edge}(i,c)\colon \prod_{j=1}^{n}\#(i-1,c,j)a\left(\sigma_1^{-1},c,c\right)\prod_{j=1}^{n}\mathcal{N}(i,c,j)a\left(\sigma_2^{-1},c,c\right)\prod_{j=1}^{n}\#(i,c,j);$$

$$\begin{aligned}\text{Vertex}(l,j)\colon \prod_{i=1}^{l-1}\Bigg[&\#\left(f_2(i,l)-1,2,j\right)\mathcal{N}\left(f_2(i,l),1,j\right)\\ &\times\prod_{c=1}^{m}a\left(j,c,f_2(i,l)\right)\#\left(f_2(i,l),1,j\right)\Bigg]\\ \times\prod_{i=l}^{k-1}\Bigg[&\#\left(f_1(l)+i-l-1,3,j\right)\prod_{c=1}^{m}a\left(j,c,f_1(l)+(i-l)\right)\\ &\times\mathcal{N}\left(f_1(l)+i-l,1,j\right)\#\left(f_1(l)+i-l,2,j\right)\Bigg];\end{aligned}$$

$$\begin{aligned}\text{Filler}(i,c)\colon\ &\#(i-1,c,n+1)\#(i-1,c,n+2)\prod_{d=1}^{m}a\left(\sigma_1^{-1},c,d\right)\prod_{j=1}^{n}\mathcal{N}(i,c,j)\\ &\times\prod_{d=1}^{m}a\left(\sigma_2^{-1},c,d\right)\#(i,c,n+1)\#(i-1,c,n+2).\end{aligned}$$

Obviously, VacI, VacII, CE(i), $\mathcal{F}(i)$, and Check(l) $(1\leqslant i\leqslant\alpha,\ 1\leqslant l\leqslant k)$, are subsequences of $\mathcal{S}$. The length of $\mathcal{S}$ is

$$\begin{aligned}|\mathcal{S}|=&\left|\prod_{i=1}^{\alpha}\left[\prod_{c=1}^{m}\left[\mathcal{X}(i,c-1)\diamondsuit_1(i,c)\right]\mathcal{X}(i,m)\right]\right|\\ &+\left|\prod_{l=1}^{k}\left[\prod_{j=n}^{1}\left[\mathcal{Y}(l,j-1)\clubsuit_1(l,j)\right]\mathcal{Y}(l,n)\right]\right|\end{aligned}$$

$$
\begin{aligned}
&+\left|\prod_{i=1}^{\alpha}\left[\prod_{c=1}^{m}\left[\mathcal{Z}(i,c-1)\spadesuit_1(i,c)\right]\mathcal{Z}(i,m)\right]\right| \\
&+\left|\mathbf{S}\prod_{i=1}^{\alpha}\left[\#(i-1)\prod_{c=1}^{m}\prod_{d=1}^{m}a\left(\tau_1^{-1}(i),c,d\right)\mathcal{N}(i)\right.\right. \\
&\qquad\left.\left.\times\prod_{c=1}^{m}\prod_{d=1}^{m}a\left(\tau_2^{-1}(i),c,d\right)\right]\#(\alpha)\mathbf{E}\,\mathrm{GlueingZone}\right| \\
&+\left|\prod_{i=1}^{\alpha}\left[\prod_{c=1}^{m}\left[\mathcal{Z}(i,c-1)\spadesuit_2(i,c)\right]\mathcal{Z}(i,m)\right]\right| \\
&+\left|\prod_{l=1}^{k}\left[\prod_{j=n}^{1}\left[\mathcal{Y}(l,j-1)\clubsuit_2(l,j)\right]\mathcal{Y}(l,n)\right]\right| \\
&+\left|\prod_{i=1}^{\alpha}\left[\prod_{c=1}^{m}\left[\mathcal{X}(i,c-1)\diamondsuit_2(i,c)\right]\mathcal{X}(i,m)\right]\right| \\
&=2\left[10+8n+m+k(4m+7-3n+nm)+k^2(8+14n+11m)\right] \\
&\quad+m\left[n+\alpha(53n+24m-4)\right].
\end{aligned}
$$

The following lemmata show that there exist a k-clique in G, if there is a common supersequence $\mathcal{S}$ of size at most M. □

Lemma 1. Vac: $\mathcal{X}\ \mathcal{Y}\ \mathcal{Z}\ \mathbf{S}\prod_{i=1}^{\alpha}\mathrm{SelectBox}(i)\#(\alpha)\mathbf{E}\mathrm{GlueingZone}\ \mathcal{Z}\ \mathcal{Y}\ \mathcal{X}$ *is subsequence of each shortest common supersequence of* Seq.

Proof. $|\mathrm{GlueingZone}| > \max\{|\mathcal{X}|, |\mathcal{Y}|, |\mathcal{Z}|\}$. □

Lemma 2. *In every shortest common supersequence of* Seq, *for each* CliqueEdge(i), $1 \leqslant i \leqslant \alpha$, *there are exactly* $(m-1)$ *of the* Edge(i,c)*-substrings,* $1 \leqslant c \leqslant m$, *in exactly one of the two* $\mathcal{X}$*-regions.*

Proof. Each $\mathcal{X}(i-1,c)$-substring of CliqueEdge(i) has exactly one matching substring in the left $\mathcal{X}$-region and exactly one in the right $\mathcal{X}$-region of each shortest common supersequence.

Let CliqueEdge(i_0) be one of the CliqueEdge(i)-sequences.

1. There cannot be more than $(m-1)$ of the m Edge(i_0,c) strings in the $\mathcal{X}$ regions, because $\prod_{i=1}^{\alpha}\mathrm{SelectBox}(i)\#(\alpha)$ and each Edge(i_0,c) contain a common subsequence of length $4m$. Therefore at least one of the Edge(i_0,c) is in the $\prod_{i=1}^{\alpha}\mathrm{SelectBox}(i)\#(\alpha)$ region of each shortest common supersequence of Seq.
2. There cannot be less than $(m-1)$ of the m Edge(i_0,c) strings in the $\mathcal{X}$ regions, because otherwise at least one of the $\mathcal{X}(i_0-1,c)$ strings of CliqueEdge(i_0) has to appear twice in the supersequence. But $|\mathcal{X}(\cdot,\cdot)| > |\mathrm{Edge}(\cdot,\cdot)|$. □

Lemma 3. *In every shortest common supersequence of* Seq, *for each* Check(l), $1 \leqslant l \leqslant k$, *there are exactly* $n-1$ *of the* Vertex(l, j)*-substrings,* $1 \leqslant j \leqslant n$, *in exactly one of the two* $\mathcal{Y}$*-regions.*

Proof. Analogous to the proof of Lemma 2. □

Lemma 4. *In every shortest common supersequence of* Seq, *for each* $\mathcal{F}(i)$, $1 \leqslant i \leqslant \alpha$, *there are exactly* $(m-1)$ *of the* Filler(i, c)*-substrings,* $1 \leqslant c \leqslant m$, *in exactly one of the two* $\mathcal{Z}$*-regions.*

Proof. Analogous to the proof of Lemma 2. □

Lemma 5. 1. *In each shortest common supersequence exactly one* Filler$(\cdot,\cdot)$*-subsequence from each* $\mathcal{F}(\cdot)$*-sequence lands between the* **S***- and* **E***-symbol.*

2. *Let* Filler(i_0, c_0) *be this subsequence. Then it lands in* SelectBox(i_0).

Proof. Follows directly from Lemma 4. □

Lemma 6. 1. *In each shortest common supersequence exactly one* Edge$(\cdot,\cdot)$*-subsequence from each* CliqueEdge$(\cdot)$*-sequence lands between the* **S***- and* **E***-symbol.*

2. *Let* Edge(i_0, c_0) *be this subsequence. Then it lands in* SelectBox(i_0).

Proof. Follows directly from Lemma 2. □

Lemma 7. 1. *In each shortest common supersequence exactly one* Vertex$(\cdot,\cdot)$*-subsequence from each* Check$(\cdot)$*-sequence lands between the* **S***- and* **E***-symbol.*

2. *Let* Vertex(l_0, j_0) *be this subsequence. Then it lands in* $\prod_{i=1}^{\alpha}$ SelectBox(i).

Proof. Follows directly from Lemma 3. □

Lemma 8. 1. *Each shortest common supersequence* $\mathcal{S}$ *is of length* $|\mathcal{S}| \geqslant M$.

2. *If* $|\mathcal{S}| = M$ *then* G *has a clique of size* k.

Proof. 1. Straightforward from the lemmata above.

2. The part of a shortest common supersequence containing $\mathbf{S}\prod_{i=1}^{\alpha}$ SelectBox$(i)\mathbf{E}$ is shortest if it has the form

$$\mathbf{S}\prod_{i=1}^{\alpha}\left[\#(i-1)\prod_{c=1}^{m}\prod_{d=1}^{m}a\big(\tau_1^{-1}(i),c,d\big)\mathcal{N}(i)\prod_{c=1}^{m}\prod_{d=1}^{m}a\big(\tau_2^{-1}(i),c,d\big)\right]\#(\alpha)\mathbf{E}$$

since then also the a-symbols representing the vertices of the graph match. □

Note, that the reduction in the proof of Theorem 2 shows $W[1]$-hardness for k-pSCT since the parameter, namely the number of objects, does not change.

Theorem 9. *k-*pSCT *is hard for* $W[1]$ *when parameterized by the number of input trees.*

5. Tractable parameterizations

Though both SCT and SCS restricted to p-objects and parameterized by the number of input objects are $W[1]$-hard, we show that the problems are fixed-parameter tractable when we allow the superobjects to be a bit longer (by parameter r) than the size of the alphabet. We define the following problems, see Table 3.

BOUNDED DUPLICATION SCS I FOR P-SEQUENCES (BDSCS I)
Input: A family of k p-sequences $x_i \in \Sigma^*$ for $i = 1, \ldots, k$ and a positive integer r representing the number of duplication events. We assume that $|\Sigma| = n$ and that each symbol of Σ occurs in at least one of the input sequences.
Parameter: (k, r).
Question: Is there a common supersequence x of length at most $n + r$?

BOUNDED DUPLICATION SCS II FOR COMPLETE P-SEQUENCES (BDSCS II)
Input: Complete p-sequences x_i over an alphabet Σ of size n, a positive integer r, and a cost function $c : \Sigma \to Z^+$.
Parameter: r.
Question: Is there a common supersequence x of duplication cost $\|x\|_c \leqslant r$ where the duplication cost is defined:

$$\|x\|_c = \sum_{a \in \Sigma} \big(n_a(x) - 1\big)c(a)$$

$n_a(x)$, $a \in \Sigma$, denotes the number of occurrences of symbol a in x.

BOUNDED DUPLICATION SCS III FOR P-SEQUENCES (BDSCS III)
Input: A family of k p-sequences $x_i \in \Sigma^*$ for $i = 1, \ldots, k$ and a positive integer r representing the number of duplication events. We assume that $|\Sigma| = n$ and that each symbol of Σ occurs in at least one of the input sequences.

Table 3
In all the problems there are k input objects, all input objects are p-objects

	Input	Parameter	Question	Complexity
pSCS				
I	$r > 0$	(k, r)	Is there a common supersequence of length at most $n + r$?	FPT
II	complete p-sequences, $r > 0$, cost function $c : \Sigma \to Z^+$	r	Is there a common supersequence of duplication cost at most r?	FPT
III	$r > 0$	r	Is there a common supersequence of length at most $n + r$?	?
pSCT				
I	complete binary p-trees, $r > 0$	(k, r)	Is there a common binary supertree of size at most $n + r$?	FPT
II	$r > 0$	(k, r)	Is there a common binary supertree of size at most $n + r$?	?
III	complete binary p-trees, $r > 0$	r	Is there a common binary supertree of size at most $n + r$?	?

Parameter: r.
Question: Is there a common supersequence x of length at most $n + r$?

BOUNDED DUPLICATION SCT FOR P-TREES (BDSCT)
Input: A family of k complete binary p-trees T_i with leaf label set Σ, $|\Sigma| = n$, and a positive integer r representing the number of duplication events.
Parameter: (k, r).
Question: Is there a common binary supertree T of the T_i of size at most $n + r$?

The definition of BOUNDED DUPLICATION SCT FOR P-TREES is reasonable for applications in the study of gene duplication events in the sense that both k and r may be small and the input trees complete when complete sequence data is available for all of the species under consideration.

We start with the complexity of the sequence problems.

Theorem 10. BOUNDED DUPLICATION SCS I FOR P-SEQUENCES *is fixed-parameter tractable.*

Proof. We describe the argument in terms of a one-person game that provides a convenient representation of a supersequence of a set of p-sequences. At each move of the game, a symbol $a \in \Sigma$ is chosen, and those sequences in the collection that have a as their first symbol are modified by deleting the first symbol. We will call this an a-move. The game is completed when all of the symbols have been deleted from all of the sequences. It is straightforward to verify that a set of p-sequences has a common supersequence x of length m if and only if the game can be completed in m moves, where the sequence of symbols chosen for the moves of the game is x. When the game is played for the set of sequences x_i we may refer to the *current* x_i, meaning the suffix of x_i that remains at an understood (current) point in the game.

Suppose the symbol a is chosen for a move of the game. For each of the sequences x_i, one of the following statements must be true:

1. The symbol a is the first symbol of the current x_i and does not otherwise occur in the current x_i.
2. The symbol a does not occur in the current x_i.
3. The symbol a occurs in the current x_i, but is not the first symbol.

If for an a-move of the game only (1) and (2) occur, we call this a *good* a-move. A move that is not good is *bad*. Our algorithm is based on the following claims (proofs straightforward and therefore left to the reader).

Claim 1. If for the input to the problem, a game is played that involves at least r bad moves, then the corresponding common supersequence x has length at least $n + r$.

Claim 2. For any yes-instance of the problem, there is a game that can be completed with at most r bad moves.

We can now describe an *FPT* algorithm based on the method of search trees [10]. By Claim 2, if the answer is "yes" then there is a game that completes with no more than r bad moves.

Algorithm.

0. The root node of the search tree is labeled with the given input.
1. A node of the search tree is expanded by making a sequence of good moves (arbitrarily) until no good move is possible. For each possible nontrivial bad move (i.e., one that results in at least one deletion), create a child node labeled with the set of sequences that result after this bad move.
2. If a node is labeled by the set of empty sequences, then answer "yes."
3. If a node has depth r in the search tree, then do not expand any further.

The correctness of the algorithm follows from Claim 2 and the following, which justifies that the sequence of good moves in (1) can be made in any order without increasing the length of the game.

Claim 3. If two different good moves, an a-move and a b-move, for $a, b \in \Sigma$, are possible at a given point in a game, then there is a completion of the game in a total of l moves including m bad moves and starting with the a-move if and only if there is a completion of the game in a total of l moves including m bad moves and starting with the b-move.

Proof. By the definition of a good move, the sequences that are current at the hypothesized point in a game can be partitioned into three subsets:

1. those that begin with a and do not otherwise contain either an a or a b,
2. those that begin with a b and do not otherwise contain either an a or a b,
3. those that do not contain either an a or a b.

Because the claim is symmetric with respect to a and b, we make the argument in only one direction. Suppose G is a game that begins with these sequences, starts with a good a-move, completes in a total of l moves, and involves at most m bad moves. The game G must eventually involve exactly one b-move that is necessarily a good move. The sequences that belong to the subset (2) are not affected by any of the moves of G until this b-move, and the b move does not affect the sequences belonging to the subsets (1) and (3) since these do not contain a b. Consequently, we can consider the game G' that is obtained from G by making the good b-move the second move (by commuting it past the prior moves of G). Thus the game G' completes in a total of at most l moves and involves at most m bad moves. The input on which G' acts is symmetric with respect to a and b, with the good a-move first, followed by the good b-move. The game that interchanges these two moves therefore completes in at most l moves and involves at most m bad moves. This proves Claim 3.

The running time of the algorithm is bounded by $O(k^r \cdot n)$. □

Corollary 9. BOUNDED DUPLICATION SCS I *is fixed-parameter tractable.*

We next consider the question of what happens to the complexity of the above problem if the parameter is considered to be just r, rather than the pair (k, r). If we restrict the input sequences to be complete, then we can show fixed-parameter tractability for a more general weighted problem. Indeed, as is frequently the case for *FPT* algorithms, solving this more general problem seems to be the key to proving tractability for the unweighted form of the problem.

Theorem 11. BOUNDED DUPLICATION SCS II FOR COMPLETE P-SEQUENCES *is fixed-parameter tractable.*

Proof. Say that two symbols $a, b \in \Sigma$ are *combinable* if ab is a substring of each x_i. We argue that if $|\Sigma| > 3r + 1$ then either there is a combinable pair of symbols or the answer is "no." Suppose the answer is "yes" and let x be any solution. Let x' be the subsequence of x consisting of those symbols of Σ that are not repeated in x. There is a natural factorization of x' into at most $2r + 1$ substrings of x. Let Σ' be the set of symbols occurring in x'. Then $|\Sigma'| > 2r + 1$ and at least one of the substrings of the factorization of x' must have length at least two. Let ab denote a substring of x of two symbols in Σ'. We argue that a and b are combinable. If not, then in at least one of the x_i we do not have the substring ab, and since the x_i are complete, one of the following must hold:

1. x_i contains the substring ba,
2. x_i contains a subsequence acb, or
3. x_i contains a subsequence bca.

It is easy to check that in each of these cases it is impossible for x to be a supersequence of x_i.

The algorithm is described as follows, based on the method of problem kernels. The argument above shows that we can efficiently determine a reason to answer "no" (if $|\Sigma| > 3r + 1$), or we can find a combinable pair ab of symbols. If we find a combinable pair, then we replace the substring ab in each x_i with a new symbol s_{ab} and give this symbol the duplication cost $c(s_{ab}) = c(a) + c(b)$. Given that the answer to this modified input is "yes" if and only if the answer for the original input is "yes," this gives us a way of reducing to a problem kernel in time $O(N^2)$ where N is the total size of the input to the problem. One direction is easy and the argument is omitted.

For the other direction, suppose x is a solution to the original instance having duplication cost at most k. We consider two cases:

(1) $n_a(x) \leqslant n_b(x)$, and
(2) $n_b(x) \leqslant n_a(x)$.

In case (1) we obtain x' from x by deleting all occurrences of b, and by replacing all occurrences of a with the symbol s_{ab}. In case (2) we obtain x' from x by deleting all occurrences of a and replacing all occurrences of b with s_{ab}. It is easy to see that the

duplication cost of x' is bounded by r. The key point is to argue that x' is a solution to the modified problem. We argue for case (1), since case (2) is essentially the same. Choose a particular subsequence y of x that is isomorphic to x_i. Then $y = y_1aby_2$, and y_1 is a subsequence of x' occurring before the relevant occurrence of a in x. Also y_2 is not affected by the deletion of b and the replacement of a with s_{ab}, so $y_1 s_{ab} y_2 = x'_i$ is a subsequence of x'. □

It would be very nice to settle the parameterized complexity of BOUNDED DUPLICATION SCS III FOR P-SEQUENCES (BDSCS III), defined in exactly the same way as BDSDS I except that the parameter is simply r rather than the pair (k, r). The proof of Theorem 1 shows the following

Theorem 12. *If* BDSCS III *is fixed-parameter tractable, then so is* DIRECTED FEEDBACK VERTEX SET (DFVS).

The interest of this theorem is that DFVS has so far resisted a number of attempts to settle its parametric complexity and is considered a major open problem [11], with most expert opinion favoring the conjecture that it is in *FPT*.

The following theorem is especially nice, since the number of duplicated leaves in a smallest common supertree presents a lower bound on the number of gene-duplication events necessary to rectify the input (gene) trees with respect to the optimal species tree when the underlying model is the duplication/loss model [15,23].

Theorem 13. BOUNDED DUPLICATION SCT FOR P-TREES *is fixed-parameter tractable.*

Proof (sketch). The algorithm is based on a combination of the methods of search trees and reduction to a problem kernel described in [10,11]. To a given tree T_i and $a \in \Sigma$, we associate an ordered partition $\pi_i(a)$ of $\Sigma - \{a\}$ in a natural way by considering the path $\rho_i(a) = (r_i = v_0, v_1, \ldots, v_s = l)$ in T_i from the root r_i or T_i to the leaf l labeled by a. Since the tree is binary, each vertex of this path other than l has another child v'_j for $j = 0, \ldots, s-1$. Let $T_i(j)$ denote the subtree rooted at v'_j and let $L_i(j)$ denote the set of leaf labels of the subtree $T_i(j)$. Then $\pi_i(a) = (L_i(0), \ldots, L_i(s-1))$.

We say that $a \in \Sigma$ is *good* if for all i, i', $1 \leqslant i, i' \leqslant k$, $\pi_i(a) = \pi_{i'}(a)$, and otherwise we say that a is *bad*. Note that if a is good then there is an obvious possibility of breaking the problem down into subproblems, since for each class of the ordered partition $\pi(a)$ each T_i yields a subtree having precisely this set of leaf labels. We will refer to this as the subproblem *induced* by the class of the ordered partition.

The algorithm proceeds by attempting to find a good $a \in \Sigma$ (this is easy by just computing and comparing the partitions), and if a good a is found, then branching in a search tree by trying all possibilities for allocating the "budget" of k duplications among the nontrivial subproblems induced by classes of $\pi(a)$. Here by *nontrivial* we mean the situation where the induced subtrees of the subproblem are not all isomorphic, so that at least one duplication event is required for a common supertree. Note that determining whether a subproblem is trivial is easy. If there are more than k nontrivial subproblems then we answer "no."

We continue in this way to build a search tree until there are no more good symbols that allow further breakdown of the problem. When the relevant subproblems are *small* (bounded in size by an appropriate function of k and r) then we answer by exhaustive search. The correctness and fixed-parameter tractability of the procedure follows from the following two claims.

Claim 1. If there is a good $a \in \Sigma$ then an optimal solution can be computed by solving the subproblems and gluing these together along the path $\rho(a)$.

The argument proving this claim first establishes inductively that an optimal solution in which a occurs without duplication can be modified into the required normal form by various surgeries that do not increase the size of the solution. A separate argument establishes that a is not duplicated in any optimal solution. This requires a similar inductive normalization by surgery that proceeds bottom up from the lower branch point of root-to-a paths in a common supertree.

Claim 2. There is a function $f(r)$ such that if $|\Sigma| > f(r)$ and $k \geqslant 2$ and all $a \in \Sigma$ are bad, then the answer is "no."

The proof of this claim makes use of Buneman's Theorem [4,5] on the reconstructibility of p-trees from triples, and on being able to find more than k disjoint sets of conflicting triples when $|\Sigma|$ is large and all $a \in \Sigma$ are bad. □

6. Summary and open problems

Our main contribution has been to explore the complexity of basic problems about sets of symbols structured by binary trees and sequences. These are related to the MAST problem, which has an important, nontrivial polynomial-time algorithm, a situation that suggests that the other three problems that are naturally similar to MAST might also be solvable in polynomial-time. One of these, the sequence analog of MAST, we have shown to be polynomial, while the problems dual to MAST concerning common supersequences and supertrees remain NP-complete. For some applications, it is natural to parameterize according to both the number of sequences or trees, and according to the number of symbols or leaves that must be duplicated in a common supersequence or supertree. Our main positive results show fixed-parameter tractability for these parameterizations for both sequences and trees. Another interesting aspect of our results are the strong parallels exhibited between the complexity of tree and sequence problems.

A number of open problems remain. In particular:

1. The fixed-parameter tractability results presented are slightly stronger for sequences. We conjecture that tree analogs of Theorems 10 and 11 should hold.
2. Does Theorem 11 still hold if the input sequences are not required to be complete? This seems to be a reasonable conjecture, but we have shown that this would imply

that DIRECTED FEEDBACK VERTEX SET is in *FPT*, also a reasonable conjecture, but apparently a difficult one [11].

3. Because in biological applications the p-sequences and p-trees are often complete, the complexity of the four basic problems LCS, SCS, LCT, and SCT for complete p-objects is an interesting issue.

Another important area for investigation concerns the practical utility of our *FPT* results. For other problems (such as VERTEX COVER) results have been encouraging in two different ways. First of all, initial *FPT* classifications have been improved several times by a variety of means. Such a trajectory of improvements seems typical for *FPT* parameterized problems. The exponential parametric running time function is now quite reasonable, with $f(k) = (1.286)^k k$ for VERTEX COVER. One would therefore expect this algorithm to be practical for k up to about 60. In fact (the second development), experiments with implementations indicate that the *FPT* algorithm is practical for k up to at least 400 (and available as a service, with applications in computational biology and other areas, on the world wide web) [13]. For the problems explored in this paper, both improved *FPT* algorithms and implementations to evaluate practical parameter ranges experimentally are appealing topics for further research.

References

[1] A.V. Aho, J.D. Ullman, Foundations of Computer Science, Freeman, 1992.
[2] H. Bodlaender, R. Downey, M. Fellows, M. Hallett, H.T. Wareham, Parameterized complexity analysis in computational biology, Comput. Appl. Biosci. 11 (1995) 49–57.
[3] H. Bodlaender, R. Downey, M. Fellows, H.T. Wareham, The parameterized complexity of the longest common subsequence problem, Theoret. Comput. Sci. A 147 (1995) 31–54.
[4] D. Bryant, Building trees, hunting for trees, and comparing trees—theory and methods in phylogenetic analysis, PhD thesis, Department of Mathematics, University of Canterbury, 1997.
[5] P. Buneman, The recovery of trees from measures of dissimilarity, in: F.R. Hodson, D.G. Kendall, P. Tautu (Eds.), Mathematics in the Archaeological and Historical Sciences, Edinburg Univ. Press, Edinburgh, 1971, pp. 387–395.
[6] L. Cai, J. Chen, R. Downey, M. Fellows, The parameterized complexity of short computation and factorization, Arch. Math. Logic 36 (1997) 321–337.
[7] J. Chen, I.A. Kanj, W. Jia, Vertex cover: further observations and further improvements, J. Algorithms 41 (2001) 280–301.
[8] R.G. Downey, M.R. Fellows, Fixed parameter tractability and completeness I: basic theory, SIAM J. Comput. 24 (1995) 873–921.
[9] R.G. Downey, M.R. Fellows, Fixed parameter tractability and completeness II: completeness for $W[1]$, Theoret. Comput. Sci. A 141 (1995) 109–131.
[10] R.G. Downey, M.R. Fellows, Parametrized computational feasibility, in: P. Clote, J. Remmel (Eds.), Feasible Mathematics II, Birkhäuser, Boston, 1995, pp. 219–244.
[11] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer-Verlag, 1998.
[12] R.G. Downey, M.R. Fellows, U. Stege, Parameterized complexity: a framework for systematically confronting computational intractability, in: F. Roberts, J. Kratochvil, J. Nešetřil (Eds.), The Future of Discrete Mathematics: Proceedings of the First DIMATIA Symposium, Stirin Castle, Czech Republic, June, 1997, in: AMS–DIMACS Proc. Ser., 1999, in print.
[13] F. Dehne, A. Rau-Chaplin, U. Stege, P.J. Taillon, Solving large FPT problems on coarse grained parallel machines, Manuscript, 2001.

[14] M. Farach, T. Przytycka, M. Thorup, On the agreement of many trees, Inform. Process. Lett. 55 (1995) 297–301.
[15] M. Fellows, M. Hallett, U. Stege, On the multiple gene duplication problem, Algorithms and Computation, in: 9th International Sympsium, ISAAC '98, in: Lecture Notes in Comput. Sci., Vol. 1533, 1998, pp. 347–356.
[16] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[17] M. Goodman, J. Czelusniak, G.W. Moore, A.E. Romero-Herrera, G. Matsuda, Fitting the gene lineage into its species lineage: a parsimony strategy illustrated by cladograms constructed from globin sequences, Syst. Zool. 28 (1979) 132–163.
[18] B. Ma, M. Li, L. Zhang, On reconstructing species trees from gene trees in term of duplications and losses, Recomb 98, in preparation.
[19] D. Maier, The complexity of some problems on subsequences and supersequences, J. ACM 25 (2) (1978) 322–336.
[20] M. Middendorf, Zur Komplexität von Einbettungsproblemen von Wortmengen, Dissertation, Fachbereich Mathematik, Universität Hannover, 1992.
[21] R.D.M. Page, Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas, Syst. Biol. 43 (1994) 58–77.
[22] T. Przytycka, Private communication, 1997.
[23] U. Stege, Resolving conflicts in problems in computational biochemistry, PhD Dissertation ETH, Zurich, 2000.
[24] U. Stege, M. Fellows, An improved fixed-parameter tractable algorithm for vertex cover, Tech. Rep., ETH Zurich, 1999.
[25] V.G. Timkovsky, Complexity of common subsequence and supersequence problems and related problems, Cybernetics 25 (1990) 1–13.
[26] R.A. Wagner, M.J. Fischer, The string-to-string correction problem, J. Assoc. Comput. Mach. 21 (1974) 168–173.