Local Search for String Problems: Brute Force is Essentially Optimal

Jiong Guo[1⋆], Danny Hermelin[2], Christian Komusiewicz[3]

[1] Universität des Saarlandes,
Campus E 1.7, D-66123 Saarbrücken, Germany.
jguo@mmci.uni-saarland.de
[2] Industrial Management and Engineering Department,
Ben-Gurion University, Beer Sheva, Israel.
hermelin@bgu.ac.il
[3] Institut für Softwaretechnik und Theoretische Informatik,
Technische Universität Berlin, D-10587 Berlin, Germany
christian.komusiewicz@tu-berlin.de

Abstract. We address the problem of whether the brute-force procedure for the local improvement step in a local search algorithm can substantially be improved when applied to classical NP-hard string problems. We examine four of the more prominent problems in this domain: Closest String, Longest Common Subsequence, Shortest Common Supersequence, and Shortest Common Superstring. Herein, we consider arguably the most fundamental string distance measure, namely the Hamming distance, which has been applied in practical local search implementations for string problems. Our results indicate that for all four problems, the brute-force algorithm cannot be considerably improved.

Key words: Local Search, Parameterized Complexity, Parameterized Intractability, Closest String, Longest Common Subsequence, Shortest Common Supersequence, Shortest Common Superstring.

1 Introduction

Local search is a universal algorithmic approach for coping with computationally hard optimization problems. It is typically applied on problems which can be formulated as finding a solution maximizing or minimizing a criterion among a number of feasible solutions. The main idea is to start with some solution, then search inside the *local neighborhood* of this solution for a better solution until a locally optimal solution has been found. The hope is then that the locally optimal solution is almost as good as a globally optimal one. See the book by Aarts and Lenstra [1] for further background and results concerning local search.

There are two main theoretical approaches to study local search: PLS-completeness [12] and parameterized local search [15, 6]. PLS-completeness can be used to show that finding a locally optimal solution is computationally hard since a lot of improvement steps might be needed until it has been found. In contrast, parameterized local search is concerned with the parameterized complexity of the problem of searching the local neighborhood of a solution in order to find a better solution. Usually the size of the neighborhood is $n^{O(k)}$, where n is the total input length, and k is a parameter measuring the "radius" of the neighborhood; that is, the maximum distance to the current solution. It is therefore natural to ask whether $n^{O(k)}$ time is required for searching this neighborhood, or whether $f(k) \cdot \text{poly}(n)$ time can be achieved. This is precisely the main question underlying the theory of *parameterized complexity* [5]. In this context, this question translates to determining whether the improvement step problem is fixed-parameter tractable or not with respect to k.

⋆ Supported by the DFG excellence cluster MMCI.

There is substantial work in parameterized local search. For example, concerning the Traveling Salesman problem, Balas [2] showed that one can find, if it exists, a better tour with "shift" distance at most k to the old one in $4^k \cdot \text{poly}(n)$ time. Marx [15] proved the non-existence of such an algorithm for the edge-exchange neighborhood. Subsequently, the complexity of local search for further neighborhood measures of Traveling Salesman was examined [10]. Fellows et al. [6] provided parameterized algorithms for local search variants of diverse graph problems such as r-Center, Vertex Cover, Odd Cycle Transversal, Max Cut, and Min-Bisection on planar graphs and proved W[1]-hardness for the general case. Fomin et al. [8] considered the Feedback Arc set in Tournaments problems and presented a subexponential-time algorithm for its edge-exchange local search version. Further, Marx and Schlotter [16] studied a variant of Stable Marriage with respect to local search. Further results concerning parameterized local search have been achieved for clustering problems [4], Boolean Constraint Satisfaction [13], and Satisfiability [18].

In this paper, we add a new realm to the study of parameterized local search by considering string problems. Stringology is one of the most widely studied areas in computer science, particularly motivated by direct applications in text mining and computational biology. Here, we consider four prominent NP-hard string problems: Closest String, Longest Common Subsequence, Shortest Common Supersequence, and Shortest Common Superstring. Local search seems to be a natural approach for dealing with string problems. For instance, a local search heuristic using the Hamming distance neighborhood has been implemented and evaluated with real-world data for a closely related variant of Closest String [7, 17].

We examine all four string problems above in the framework of parameterized local search. Herein, we consider the Hamming distance neighborhood of a temporary solution and prove that the local search version of all these problems are W[1]-hard even on alphabets of constant size, with the Hamming distance k between the old and new solutions as parameter. Since the Hamming distance seems to be the most simple distance between strings, our results could serve as the basis for proving the hardness for other distance neighborhoods. Moreover, for all problems except Shortest Common Supersequence, we can exclude the existence of algorithms with running-times $n^{o(k)}$. Thus, for these three problems, the $n^{O(k)}$-time brute-force cannot be substantially improved and is essentially optimal.

2 Preliminaries

For a string S we write $|S|$ to denote the *length* of S. We use $S[i]$, $1 \leq i \leq |S|$, to denote the letter at position i in S and use $S[i, j]$, $1 \leq i < j \leq |S|$, to denote the *substring* $S[i] \cdots S[j]$ of S from position i to position j. A substring of the form $S[i, n]$ is called a *suffix* of S, and a substring $S[1, j]$ is called a *prefix*. For a given suffix T of S, we write $S - T$ to denote the string $S[1, |S| - |T|]$. We use $S-$ as a shorthand for $S - S[|S|]$. A string T is a *subsequence* of S if T can be obtained from S by deleting some letters; that is, if there exists a sequence of positions $i_1 < \cdots < i_{|T|}$ with $S[i_j] = T[j]$ for all $j \in \{1, \ldots, |T|\}$. If T is a subsequence of S, then S is called a *supersequence* of T. The *Hamming distance* $d_H(S, T) := |\{i : S[i] \neq T[i]\}|$ between two string S and T of equal length is defined as the number of positions in which the two strings differ. We define the Hamming distance of a string S to a set $\mathcal{T}$ of strings as $d_H(S, \mathcal{T}) := \max_{T \in \mathcal{T}} d_H(S, T)$.

We will analyze our local search string problems using the framework of parameterized complexity [5]. In parameterized complexity, problem instances are ordered pairs $(x, k) \in \{0, 1\}^* \times \mathbb{N}$, where x is a binary string that encodes the combinatorial input of the problem (in our case, a set of strings and a few integers), and k is a non-negative integer which is referred to as the *parameter*.

For this paper, the only notion from parameterized complexity theory is the concept of reduction: A *parameterized reduction* from a parameterized problem L to another parameterized problem L' is an algorithm with running time $f(k) \cdot \text{poly}(|x|)$ for some computable $f()$, that maps an instance (x, k) of L to an instance (x', k') of L' such that:

(i) $k' \leq g(k)$ for some computable $g()$, and
(ii) $(x, k) \in L \iff (x', k') \in L'$.

Note that the running-time of the reduction is allowed to be super-polynomial in k, yet the exponent of the polynomial which depends on n in this expression is required to be independent of k. Thus, a running-time of *e.g.* $O(2^k \cdot |x|^5)$ is allowed, while $O(|x|^k)$ is not. If $g()$ is linearly bounded, *i.e.* $g(k) = O(k)$, then we say that such a reduction is a *linear parameterized reduction.*

Parameterized intractability is defined via circuit satisfiability problems. A constant-depth circuit is said to have *weft* t if the maximum number of gates with unbounded fan-in on any input-output path is t. The parameterized Weft-t Weighted SAT(k) problem asks to determine, given a circuit of weft t and a parameter k, whether the circuit can be satisfied by an assignment of Hamming weight k (*i.e.* by setting k of its input gates to 1). For every $t \geq 1$, the class W[t] is defined as the set of all problems reducible, via a (not necessarily linear) parameterized reduction, to the Weft-t Weighted SAT(k) problem. It is not difficult to show that W[1] $\subseteq$ W[2] ..., and it is widely believed that all these inclusions are proper. In this paper we will only be concerned with W[1] and W[2]. If there is a parameterized reduction from a W[1]-hard (W[2]-hard) problem to a parameterized problem L, then this implies W[1]-hardness (W[2]-hardness) of L.

The hardness results in this paper are obtained by parameterized reductions from the following three well-known problems which all have the solution size k as parameter: In the W[2]-hard Multicolored Hitting Set(k) (MHS(k)) problem, the input is a hypergraph $(V, \mathcal{E})$ and a coloring function $c : V \to \{c_1, \ldots, c_k\}$ which assigns one of k colors to each vertex in V. The goal is to determine whether there exists a size-k subset $H \subseteq V$ with $H \cap E \neq \emptyset$ for all $E \in \mathcal{E}$, such that H is *multicolored*, that is, $|\{v \in H : c(v) = c_i\}| = 1$ for all $c_i \in \{c_1, \ldots, c_k\}$. In the W[1]-hard Multicolored Independent Set(k) (MIS(k)) problem, the input is a graph (V, E) and a coloring function $c : V \to \{c_1, \ldots, c_k\}$, and the goal is to determine if (V, E) has a multicolored independent set $I \subseteq V$. The W[1]-hard Multicolored Clique(k) (MC(k)) is defined similarly, except the goal is to determine the existence of a multicolored clique instead of a multicolored independent set.

While parameterized reductions can show hardness in the parameterized sense (*e.g.* W[1]-hardness or W[2]-hardness), a well-known result of Chen *et al.* [3] shows that linear reductions can be used to obtain seemingly stronger results related to hardness in the classical world. This is done via the so-call *Exponential Time Hypothesis* (ETH) of Impagliazzo and Paturi [11] which states that the 3-SAT problem cannot be solved in $2^{o(n)}$ time. The most relevant part of the paper by Chen *et al.* [3] to our work is stated in the lemma below. We refer the reader to [14] for a recent survey on ETH-based running time lower bounds.

Lemma 1. *Let L be a parameterized problem with parameter k, and assume that there is a linear parameterized reduction from either* MHS(k)[1]*,* MIS(k)*, or* MC(k) *to L. Then unless ETH fails, L cannot be solved by an algorithm running in $n^{o(k)}$ time.*

[1] For MHS(k), Chen *et al.* [3] do not explicitly make this statement, but it can be inferred using the standard simple reduction from Dominating Set

3 Closest String

The first local search string problem we consider is a local search variant of the Closest String problem. Let Σ denote some arbitrary alphabet, and n be a positive integer. In Closest String, the input is a set $\mathcal{T} \subseteq \Sigma^n$ of strings and an integer d, and the goal is to determine whether there is a string $S \in \Sigma^n$ such that $d_H(S, \mathcal{T}) \leq d$. The local search variant of this problem that we consider is defined as follows:

Local Search Closest String (LSCS):
Input: A set $\mathcal{T} := \{T_1, \ldots, T_m\} \subseteq \Sigma^n$ of input strings, a temporary solution string $S \in \Sigma^n$ with $d := d_H(S, \mathcal{T})$, and a nonnegative integer k.
Question: Is there a string $\widetilde{S}$ of length n such that $d_H(\widetilde{S}, \mathcal{T}) < d$ and $d_H(\widetilde{S}, S) \leq k$?

Thus, we are given a temporary solution string S, and we want to find a better solution $\widetilde{S}$ in the k-neighborhood of S, where this neighborhood is defined w.r.t. Hamming distance.

We denote the different parameterizations of this problem by appending the parameters to the problem name in parenthesis. Thus, LSCS(k) for instance, is the LSCS problem parameterized by k. Observe that LSCS can be solved by a brute-force algorithm in $O(n^{k+1} \cdot m)$ time. It is also not difficult to devise a bounded search tree algorithm with running-time $d^k \cdot poly(n, m)$ using the following observation: As long as S (or an intermediate solution) differs from some input string in at least d positions, then one of these positions in S must be changed. Achieving a $f(m) \cdot \text{poly}(n)$-time algorithm by modifying the Integer Linear Programming-based algorithm of Gramm *et al.* [9] is also possible. Below, we will show that for the parameter k, one cannot substantially improve on the brute-force algorithm in general, even in the case where the strings are binary.

Theorem 1. *There is a linear parameterized reduction from* MHS(k) *to* LSCS($k + |\Sigma|$).

Proof. Given an instance $(V := \{1, \ldots, |V|\}, \mathcal{E}, c)$ of MHS(k) with $c : V \to \{c_1, \ldots, c_k\}$, we create an instance $(\mathcal{T}, S, k)$ of LSCS($k, |\Sigma|$) as follows: For each $E \in \mathcal{E}$, we create a string T_E of length $|V|$ for which $T_E[v] := c(v)$ for all $v \in E$, and $T_E[v] := 0$ for all $v \in V \setminus E$. For each $c \in \{c_1, \ldots, c_k\}$, we construct a string T_c with $T_c = c^{|V|}$. The set of strings $\mathcal{T}$ is then defined as $\mathcal{T} := \{T_E : E \in \mathcal{E}\} \cup \{T_c : c \in \{c_1, \ldots, c_k\}\}$, and the string S is set to $S := 1^{|V|}$. Thus, the input strings are over the $k + 2$ letter alphabet $\Sigma := \{0, 1, c_1, \ldots, c_k\}$, and $d_H(S, \mathcal{T}) = |V|$.

Cleary, the above construction can be carried out in polynomial time. To see its correctness, observe that given a multicolored hitting set $H \subseteq V$ of size k for $(V, \mathcal{E})$, we can construct a solution string $\widetilde{S}$ for $(\mathcal{T}, S, k)$ by setting $\widetilde{S}[v] = c(v)$ for all $v \in H$, and $\widetilde{S}[v] = 1$ for all $v \in V \setminus H$. Indeed, $d_H(S, \widetilde{S}) = k$, and it is not difficult to verify that $d_H(\widetilde{S}, T) < |V| = d_H(S, \mathcal{T})$ for every $T \in \mathcal{T}$. Conversely, a solution string $\widetilde{S}$ for $(\mathcal{T}, S, k)$ must include at least one occurrence of the letter c for every color $c \in \{c_1, \ldots, c_k\}$, since otherwise $d_H(\widetilde{S}, T_c) = |V|$. Since $d_H(S, \widetilde{S}) \leq k$, this implies that $\widetilde{S}$ must include exactly one such occurrence for each $c \in \{c_1, \ldots, c_k\}$. Furthermore, for every $E \in \mathcal{E}$ we have $d_H(\widetilde{S}, T_E) < |V|$, which implies that there is a position $v \in V$ for which $\widetilde{S}[v] = T_E[v]$, which in turn implies that $v \in E$. It therefore follows that the set $H := \{v : \widetilde{S}[v] \neq 1\}$ is a multicolored hitting set of size k for $(V, \mathcal{E})$. □

We now modify the construction in the proof of Theorem 1 so that the input strings are all over the binary alphabet $\Sigma := \{0, 1\}$. The difficulty for inputs with a binary alphabet is that flipping a character from 0 to 1 not only means "moving towards" the input strings with a 1 at this position, but also "moving away" from all other input strings. Let $(V := \{1, \ldots, |V|\}, \mathcal{E}, c)$ be an instance of MHS(k), and assume w.l.o.g. that $|E| \leq |V| - k$ for each $E \in \mathcal{E}$. Set the individual input string

length to $n := |V| + |V| \cdot |\mathcal{E}| + 2^k \cdot |V|$, and set the temporary solution S to 0^n. For each $E \in \mathcal{E}$ create a string T_E of length n. For each $v \in \{1, \ldots, |V|\}$, set $T_E[v] := 1$ if $v \in E$ and $T_E[v] := 0$ otherwise. Note that the Hamming distance between $T_E[1, |V|]$ and $S[1, |V|]$ is exactly $|E| \leq |V| - k$. The remaining positions are used to "pad" the distance between T_E and S to $|V| - k$. To this end, assign a unique number $i \in \{1, \ldots, |\mathcal{E}|\}$, and use the substring $T_E[i \cdot |V| + 1, (i+1) \cdot |V|]$ to pad the distance between T_E and S; that is, set the first $|V| - k - |E|$ positions in this substring to 1 and all other positions in $T_E[|V| + 1, n]$ to 0.

Next, add an additional set of strings which enforce that for each proper subset of colors $C \subset \{c_1, \ldots, c_k\}$, the set of colors used by a solution string $C(\widetilde{S}) := \{c(v) : \widetilde{S}[v] = 1\}$ is not C. Since we enforce this for each proper subset, it will follow that $C(\widetilde{S}) := \{c_1, \ldots, c_k\}$. For each proper $C \subset \{c_1, \ldots, c_k\}$, construct a string T_C such that, for each $v \in \{1, \ldots, |V|\}$, we have $T_C[v] = 0$ if $c(v) \in C$ and $T_C[v] = 1$ otherwise. Note that the distance between $S[1, |V|]$ and $T_C[1, |V|]$ equals the number of vertices in V not colored by a color in C. Pad the distance between T_C and S to $|V| - |C|$ by assigning T_C a unique number $i \in \{1, \ldots, 2^k - 1\}$, and let x denote the number of positions v in $T_C[1, |V|]$ with $T_C[v] = 0$. Note that $x \geq |C|$ since for each color $c \in C$ there is at least one vertex colored c. Consequently, set the first $x - |C|$ positions in $T_C[|V| \cdot (|\mathcal{E}| + i) + 1, |V| \cdot (|\mathcal{E}| + i + 1)]$ to 1, and all remaining unspecified positions to 0. Observe that in this way $T_\emptyset = 1^{|V|}0^{n-|V|}$.

This concludes the construction of the set $\mathcal{T}$ of input strings, and the instance $(\mathcal{T}, S, k)$ of LSCS(k). Clearly this construction can be performed in $2^k \cdot \text{poly}(n, m)$ time, and therefore it is a parameterized reduction. Furthermore, observe that $d_H(S, \mathcal{T}) = |V|$, and that this distance is obtained by the distance between S and $T_\emptyset$. We prove the correctness of our reduction using the following two lemmas.

Example 1. Suppose the hypergraph in the input instance to MHS(k) is given by $V = \{1, 2, 3, 4\}$ and $\mathcal{E} = \{E_1 = \{2, 3\}, E_2 = \{1\}, E_3 = \{2, 4\}, E_4 = \{1, 3\}\}$. Also assume that $k = 2$, and vertices $\{2, 3\}$ are colored r (for red), and vertices $\{1, 4\}$ are colored b (for blue). The set of strings $\mathcal{T}$ is constructed as:

- $T_1 =$ 0110 0000000000000000000000000000
- $T_2 =$ 1000 0000100000000000000000000000
- $T_3 =$ 0101 0000000000000000000000000000
- $T_4 =$ 1010 0000000000000000000000000000
- $T_\emptyset =$ 1111 0000000000000000000000000000
- $T_{\{r\}} =$ 1001 0000000000000000000010000000
- $T_{\{b\}} =$ 0110 0000000000000000000000001000

For readability purposes, we inserted above an artificial gap between the prefix and padding-suffix of each string.

Lemma 2. *If* $(V, \mathcal{E}, c) \in$ MHS(k) *then* $(\mathcal{T}, S, k) \in$ LSCS(k).

Proof. Let $H \subseteq V$ be a multicolored hitting set of size k for $(V, \mathcal{E})$. Consider the string $\widetilde{S}$ with $\widetilde{S}[v] = 1$ if $v \in H$ and $\widetilde{S}[v] = 0$ otherwise. Clearly, $d_H(S, \widetilde{S}) = k$. We show that $d_H(\mathcal{S}, \widetilde{S}) < |V|$; that is, that $\widetilde{S}$ is a solution to the instance $(\mathcal{T}, S, k)$. First, consider an arbitrary input string $T_E \in \mathcal{T}$ corresponding to $E \in \mathcal{E}$. Since $d_H(T_E, S) = |V| - k$ and $d_H(S, \widetilde{S}) = k$, the distance between $\widetilde{S}$ and T_E can be at most $|V|$. This is the case if T_E has a 0 at each position v with $\widetilde{S}[v] = 1$. Since H is a hitting set, there is a vertex $v \in H$ with $v \in E$, and for this position v we have $\widetilde{S}[v] = T_E[v] = 1$. Hence, $d_H(T_E, \widetilde{S}) < |V|$.

Next, consider an arbitrary string $T_C \in \mathcal{T}$ corresponding to a proper subset $C \subset \{c_1, \ldots, c_k\}$. Clearly $d_H(T_\emptyset, \widetilde{S}) = |V| - k < |V|$. Assume thus that $C \neq \emptyset$. By construction, $d_H(S, T_C) = |V| - |C|$.

As H is multicolored and of size k, for any color $c \in \{c_1 \ldots, c_k\}$ there is exactly one $v \in H$ with $c(v) = c$ for which $0 = S[v] \neq \widetilde{S}[v] = 1$. Thus, restricted to the positions that correspond to a color $c \in C$, the distance between $\widetilde{S}$ and T_C is increased by 1 compared to distance between $\widetilde{S}$ and T_C, since $T_C[v] = 0$ for all such positions v. For a color $c \notin C$, this distance is decreased by 1, as $T_C[v] = 1$ for all such positions $v \in \{1, \ldots, |V|\}$ with $c(v) = c$. Since C is a proper subset of $\{c_1, \ldots, c_k\}$, it follows that the distance between T_C and $\widetilde{S}$ is at most $|V| - |C| + |C| - 1 < |V|$. □

Lemma 3. *If* $(\mathcal{T}, S, k) \in$ LSCS(k) *then* $(V, \mathcal{E}, k) \in$ MHS(k).

Proof. Let $\widetilde{S}$ be a solution string for $(\mathcal{T}, S, k)$ with $d_H(S, \mathcal{T}) < |V|$ and $d_H(\widetilde{S}, S) \leq k$. Then the set of positions $H := \{v \in \{1, \ldots, n\} : \widetilde{S}[v] = 1\}$ which are set to 1 in $\widetilde{S}$ is of size at most k. Since for each $i \in \{|V| + 1, n\}$ there is at most one string $T \in \mathcal{T}$ with $T[i] = 1$, we can also assume that $H \subseteq V$, since any 1 in the substring $\widetilde{S}[|V| + 1, n]$ can be matched only to a single $T \in \mathcal{T}$, and can thus be shifted to the substring $\widetilde{S}[1, |V|]$ while making sure that the distance from T is less than $|V|$. (Note that if $T[1, |V|] = \widetilde{S}[1, |V|]$, then replacing all 1's with 0's in $\widetilde{S}[|V| + 1, n]$ always results in a string of distance less than $|V|$ to T.) We next argue that H is a multicolored subset of V, and has size precisely k.

Assume not. Then the set $C := \{c(v) : v \in H\}$ of colors in H is a proper subset of $\{c_1, \ldots, c_k\}$, and so consider the string $T_C \in \mathcal{T}$. By construction of T_C, it follows that the difference $d_H(S, T_C) - d_H(\widetilde{S}, T_C)$ is decreased by 1 for every $v \in H$ with $c(v) \in C$, and increased by 1 for every $v \in H$ with $c(v) \notin C$. Since the color set of H is precisely C, we only have vertices of the first type, and so

$$d_H(\widetilde{S}, T_C) = d_H(S, T_C) + |H| = |V| - |C| + |H| \geq |V|,$$

contradicting the assumption that $d_H(\widetilde{S}, \mathcal{T}) < |V|$.

Thus, we can assume that $H \subseteq V$ is multicolored, and has precisely k elements. Furthermore, H is also a hitting set for $(V, \mathcal{E})$: Since $d_H(\widetilde{S}, T_E) < |V|$ for each T_E, there is at least one position $v \in H$ with $T_E[v] = 1$ (otherwise, the distance between T_E and $\widetilde{S}$ is at least $|V|$). □

The two lemmas above combined prove the correctness of our reduction. We therefore have the following theorem.

Theorem 2. *There is a linear parameteric reduction from* MHS(k) *to* LSCS(k) *for binary strings.*

By combining Lemma 1 and Theorem 2, we obtain the following lower bound for LSCS restricted to binary strings.

Corollary 1. LSCS(k) *restricted to binary strings is* W[2]*-hard. Moreover, the problem has no* $n^{o(k)}$ *algorithm unless ETH fails.*

4 Longest Common Subsequence

The Longest common subsequence (LCS) problem asks to determine whether an input set $\mathcal{T}$ of strings has a string S of some specified length ℓ such that S is a subsequence of each string $T \in \mathcal{T}$. In this section we consider the following local search variant of this problem:

Local Search Longest Common Subsequence (LSLCS):
Input: A set $\mathcal{T} := \{T_1, \ldots, T_m\}$ of input strings over an alphabet Σ, a temporary solution string S such that S is a subsequence of each string in $\mathcal{T}$, and a nonnegative integer k.
Question: Is there a letter $\sigma \in \Sigma$ and a string $\widetilde{S}$ of length $|S|$ such that $\widetilde{S}\sigma$ is a subsequence of each string in $\mathcal{T}$ and $d_H(\widetilde{S}, S) \leq k$?

Observe that LSLCS can be solved in $n^{O(k)}$ time by brute-force. We show that it is unlikely to substantially improve on this algorithm, even in the case of constant-size alphabets. As a warm-up, we begin with the very easy case of unbounded alphabets.

Lemma 4. *There is a linear parameterized reduction from* LCS(ℓ) *to* LSLCS(k).

Proof. Let $\mathcal{T} := \{T_1, \ldots, T_m\}$ be an input set of strings to the LCS(ℓ) problem. We create an instance for LSLCS(k) by setting $S := \$^{\ell-1}$, where \$ is a letter not appearing in any string of $\mathcal{T}$, and then setting $\mathcal{T}' := \{ST_1, T_2S, \ldots, T_mS\}$. It can easily be verifed to see that $(\mathcal{T}', S)$ has a solution $\widetilde{S}\sigma$ with $d_H(\widetilde{S}, S) \leq k = \ell - 1$ iff $\mathcal{T}$ has a common subsequence of length ℓ. □

We next proceed to the more involved case where $|\Sigma|$ is part of the parameter. We present a reduction from MIS(k) to LSLCS($k + |\Sigma|$). Let $(G = (V, E), c)$ denote an instance of MIS(k), where G is a graph and c is a coloring function $c : V \to \{c_1 \ldots, c_k\}$. By padding (G, c), we can assume w.l.o.g, that each color class in G has precisely n vertices, that is, $|\{v : c(v) = c_i\}| = n$ for each $i \in \{1, \ldots, k\}$. The general idea of the reduction is to construct an instance $(\mathcal{T}, S, k' = k)$ of LSLCS($k + |\Sigma|$), with two *enforcement strings* $T_1, T_2 \in \mathcal{T}$, such that adding any letter at the end of S forces modifications in S that correspond to the selection of k vertices of different colors in G. We then add edge strings T_e to $\mathcal{T}$ corresponding to edges $e \in E$ in order to ensure that these selected vertices form an independent set in G. The complete details are given below.

We begin by describing the solution string S. The string S consists of a suffix $S^* := (\$\pounds^{k+1}\$)^{k+1}$, where \$ and $\pounds$ are two letters of the alphabet that do not appear elsewhere in S. The prefix of S consists of k substrings, or blocks, one for each color class. The substring $S(c_i)$ corresponding to c_i is defined as the string $S(c_i) := \overrightarrow{c_i}(0\#)^n\overleftarrow{c_i}$. The whole string S is thus constructed as

$$S := S(c_1) \cdots S(c_k)S^*.$$

Next we construct the two enforcement strings $T_1, T_2 \in \mathcal{T}$. The string T_1 contains the string S as its suffix. Its prefix contains k blocks, one for each color class of G, where the i'th block $T_1(c_i)$ is defined as $T_1(c_i) := \overrightarrow{c_i}(0\#1\#)^{n-1}\overleftarrow{c_i}$. The prefix of T_1 is separated from its suffix with the string S^* to form the string

$$T_1 := T_1(c_1) \cdots T_1(c_k)S^*S.$$

The string T_2 also contains k blocks, each corresponding to a color of G, where the block corresponding to c_i is constructed as $T_2(c_i) := \overrightarrow{c_i}(01\#)^n\overleftarrow{c_i}$. We concatenate all these blocks with the suffix $S^*\$$ to obtain the string

$$T_2 := T_2(c_1) \cdots T_2(c_k)S^*\$.$$

Finally, for each edge $e \in E$, we construct an input string T_e as follows. Assume that the vertices in each color class are ordered. Let e be an edge between the x'th vertex of color c_i and the y'th vertex of color c_j, where $i < j$. The string T_e consists of two blocks for each color class of G, defined by

- $T_e^1(c_i) := \overrightarrow{c_i}(01\#)^{x-1}0\#(01\#)^{n-x}\overleftarrow{c_i}$,
- $T_e^2(c_j) := \overrightarrow{c_j}(01\#)^{y-1}0\#(01\#)^{n-y}\overleftarrow{c_j}$,
- $T_e^2(c_i) := \overrightarrow{c_i}(01\#)^n\overleftarrow{c_i}$,
- $T_e^1(c_j) := \overrightarrow{c_j}(01\#)^n\overleftarrow{c_j}$,
- $T_e^1(c_\ell) := T_e^2(c_\ell) := \overrightarrow{c_\ell}(01\#)^n\overleftarrow{c_\ell}$, for all $\ell \neq i, j$.

We then construct T_e by concatenating all these blocks, along with the suffix $S^*\$$ to form the string

$$T_e := T_e^1(c_1)\cdots T_e^1(c_k)T_e^2(c_1)\cdots T_e^2(c_k)S^*\$.$$

Setting $\mathcal{T} := \{T_1, T_2\} \cup \{T_e : e \in E\}$ completes the construction of our LSLCS($k+|\Sigma|$) instance $(\mathcal{T}, S, k)$. Observe that S is indeed a subsequence of all strings in $\mathcal{T}$, and that Σ is an alphabet of size $2k+5$ consisting of the letters $\overrightarrow{c_1}$, $\overleftarrow{c_1}$, ..., $\overrightarrow{c_k}$, $\overleftarrow{c_k}$, 0, 1, #, \$, and £. We now make two observations that lead to the soundness and completeness of our reduction.

Lemma 5. *Suppose that $\widetilde{S}\sigma$ is a solution string for the constructed instance $(\mathcal{T}, S, k)$. Then $\widetilde{S}\sigma = \widetilde{S}(c_1)\cdots\widetilde{S}(c_k)S^*\sigma$, where for each $i \in \{1, \ldots, k\}$, the substring $\widetilde{S}(c_i)$ is obtained from $S(c_i)$ by replacing exactly one occurrence of the letter 0 with the letter 1.*

Proof. Observe that no matter what σ is chosen to be, the string $S^*\sigma$ is not a subsequence of S. Furthermore, it is not difficult to see that modifying any k letters in $S^*\sigma$ results in a string which is still not a subsequence of S. Since $\widetilde{S}\sigma$ is a subsequence of T_1, this means that the suffix of length $|S^*|+1$ in $\widetilde{S}\sigma$ must be a subsequence of S^*S. Thus, the remaining prefix of $\widetilde{S}\sigma$ must be a subsequence of $T_1(c_1)\cdots T_1(c_k)$. But as each $T_1(c_i)$ contains one less occurrence of the letter 0 than $S(c_i)$, the only way for this prefix to be a subsequence of $T_1(c_1)\cdots T_1(c_k)$ is if $\widetilde{S}\sigma$ can be written as $\widetilde{S}(c_1)\cdots\widetilde{S}(c_k)S^*\sigma$, where each $\widetilde{S}(c_i)$ is obtained from $S(c_i)$ by replacing exactly one occurrence of the letter 0 in $S(c_i)$. Clearly, the only two possibilities are to replace the 0 with either 1 or #, since otherwise $\widetilde{S}(c_i)$ would not be a subsequence of $T_1(c_i)$. However, since $\widetilde{S}\sigma$ is also a subsequence of T_2, the letter # cannot be chosen since T_2 does not contain the subsequence $\overrightarrow{c_i}\#^{n+1}\overleftarrow{c_i}$. □

According to Lemma 5 above, we can think of the positions in which $\widetilde{S}(c_1)\cdots\widetilde{S}(c_k)$ differs from $S(c_1)\cdots S(c_k)$ as an encoding the selection of k vertices, one for each color class of G. We refer to these vertices as the *set of vertices selected by* $\widetilde{S}$.

Lemma 6. *The set $I \subseteq V(G)$ of vertices selected by $\widetilde{S}$ is a multicolored independent set in G.*

Proof. Suppose that I contains two vertices which are adjacent by an edge e in G, and assume that these vertices are the x'th vertex of color class c_i and the y'th vertex of color class c_j. Then by Lemma 5, the solution string $\widetilde{S}\sigma$ contains the subsequence

$$\overrightarrow{c_i}(0\#)^{x-1}1\#(0\#)^{n-x}\overleftarrow{c_i}\,\overrightarrow{c_j}(0\#)^{y-1}1\#(0\#)^{n-y}\overleftarrow{c_j}.$$

However, this string is not a subsequence of $T_e \in \mathcal{T}$, a contradiction. □

Lemma 6 implies that if $(\mathcal{T}, S, k)$ has a solution string $\widetilde{S}\sigma$ with $d_H(\widetilde{S}, S) \leq k$ then G has a multicolored independent set of size k. Conversely, if G has a multicolored independent set I of size k, then it can readily be verified that by choosing $\widetilde{S}$ such that the k vertices it selects are precisely I, we have a solution string $\widetilde{S}\$$ for $(\mathcal{T}, S, k)$. Thus, since our construction can be carried out in polynomial time, and since MIS(k) is W[1]-complete, we obtain Theorem 3 below.

Theorem 3. *There is a linear parameterized reduction from* MIS(k) *to* LSLCS($k+|\Sigma|$).

We next sketch how to reduce the alphabet in our construction to constant size. For each $i \in \{1, \ldots, k\}$, replace the letters $\overrightarrow{c_i}$ and $\overleftarrow{c_i}$ with the substrings $p^{\alpha(i)}$ and $q^{\alpha(i)}$ respectively, where $\alpha(k) := 1$ and $\alpha(i) := \alpha(k) + \cdots + \alpha(i+1) + 1$ for $i < k$. The new alphabet is of size 7. It is not difficult to verify that Lemma 5 still holds under this modification. The rest of the proof remains unchanged.

Corollary 2. LSLCS(k) *restricted to strings over a constant-size alphabet is* W[1]*-hard. Moreover, the problem has no $n^{o(k)}$ algorithm unless ETH fails.*

5 Shortest Common Supersequence

In this section, we consider a local search version of Shortest Common Supersequence (SCSeq). In SCSeq, the input is a set of strings $\mathcal{T}$ and an integer ℓ, and the question is whether there exists a string S of length ℓ which is a supersequence of all strings in $\mathcal{T}$. The local search variant of this problem that we consider is given by:

Local Search Shortest Common Supersequence (LSSCSeq):
Input: A set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of strings over an alphabet Σ, a string S which is a supersequence of all T_i's, and a positive integer k.
Question: Is there a string $\widetilde{S}$ of length $|S| - 1$ which is a supersequence of all T_i's such that $d_H(S-, \widetilde{S}) \leq k$?

In other words, the new solution supersequence $\widetilde{S}$ is created from S by removing the last position of S and modifying at most k remaining positions. The main result of this section is the theorem below.

Theorem 4. *There is a linear parameterized reduction from* MIS(k) *to* LSSCSeq(k) *restricted to strings over an alphabet of constant size.*

Let $(G = (V, E), c)$ denote an arbitrary input of MIS(k) with $c : V \to \{c_1, \ldots, c_k\}$. We assume w.l.o.g. that there are n vertices colored c_i, for each color $c_i \in \{c_1, \ldots, c_k\}$, and that any pair of vertices with equal color are adjacent in G. Furthermore, to ease our presentation, we assume that the edges in G are directed; that is, E contains the two ordered pairs (u, v) and (v, u) for every pair of adjacent vertices u and v in G.

In our construction, we use the first part of the supersequence S to encode the k color classes of the vertex set of G. The main idea of the reduction is to construct an enforcement string T_1 which is a suffix of S and, after the removal of the last letter of S, can only be matched to the first part of S. This match forces that three positions of each color class are changed in the first part of S. These positions correspond to the vertex of this color class that is in the independent set. The remaining input strings are used to encode the edges of G, and to force that the changed positions of each color class represent the same vertex from this class. For presentation purposes, we construct strings over an alphabet of size $O(k)$. The reduction could be adapted to the constant alphabet case in a similar way as is done in Section 4. The details are as follows.

We begin by constructing the temporary solution S. First we create three substrings for each color $c_i \in \{c_1, \ldots, c_k\}$, which we refer to as *selection blocks*:

- $S^1(c_i) := \overrightarrow{c_i}(01\#)^n\overleftarrow{c_i}$,
- $S^2(c_i) := \overrightarrow{c_i}(00\#)^n\overleftarrow{c_i}$, and
- $S^3(c_i) := \overrightarrow{c_i}(01\#)^n\overleftarrow{c_i}$.

We construct S by concatenating the selection blocks, using the letter & to separate the three sets of selection blocks. We then add a suffix to S: The string $S^* := (\$\pounds^n\$)^{k+1}$ concatenated to the input string $T_1 \in \mathcal{T}$ which will be specified later. The string S is then given by

$$S := S^1(c_1) \cdots S^1(c_k)\,\&\,S^2(c_1) \cdots S^2(c_k)\,\&\,S^3(c_1) \cdots S^3(c_k)\,S^*\,T_1.$$

Next, we construct the input string T_1 which is the first of two input strings that will act as enforcement strings, enforcing the changes in S to occur in its selection blocks in a controlled fashion. For $c_i \in \{c_1, \ldots, c_k\}$, define

– $T_1^1(c_i) := \overrightarrow{c_i}\, 0^{n+1}\, \overleftarrow{c_i}$,
– $T_1^2(c_i) := \overrightarrow{c_i}\, 1\, \overleftarrow{c_i}$, and
– $T_1^3(c_i) := \overrightarrow{c_i}\, 0^{n+1}\, \overleftarrow{c_i}$.

We construct T_1 using these substrings, the separation letter &, and the suffix S^*:

$$T_1 := T_1^1(c_1) \cdots T_1^1(c_k) \,\&\, T_1^2(c_1) \cdots T_1^2(c_k) \,\&\, T_1^3(c_1) \cdots T_1^3(c_k)\, S^*.$$

The second enforcement string T_2 is constructed using the following substrings corresponding to a color $c_i \in \{c_1, \ldots, c_k\}$:

– $T_2^1(c_i) := \overrightarrow{c_i}\, (0\#)^n\, \overleftarrow{c_i}$,
– $T_2^2(c_i) := \overrightarrow{c_i}\, (0\#)^n\, \overleftarrow{c_i}$, and
– $T_2^3(c_i) := \overrightarrow{c_i}\, (0\#)^n\, \overleftarrow{c_i}$.

The string T_2 is then constructed as

$$T_2 := T_2^1(c_1) \cdots T_2^1(c_k) \,\&\, T_2^2(c_1) \cdots T_2^2(c_k) \,\&\, T_2^3(c_1) \cdots T_2^3(c_k)\, S^*\, T_1 -\, .$$

To complete the construction of $\mathcal{T}$, we construct a string T_e for each $e \in E$. These strings are composed of substrings that correspond to vertices of G. Let $v \in V$ with $c(v) := c_i$, and assume v is the x'th vertex of color c_i. The string $T(v)$ is defined by

$$T(v) := \overrightarrow{c_i}\, (0\#)^{x-1}\, 01\, (0\#)^{n-x}\, \overleftarrow{c_i}.$$

The string T_e is constructed as $T_e := T(u) \,\&\, T(v)$ if $e := (u, v)$ (recall that we assume that the edges are directed, and that any pair of vertices with the same color are adjacent).

To finalize our construction, we set the parameter k' of the LSSCSEQ instance to $3k$. Clearly the instance $(\mathcal{T}, S, k')$ can be constructed in polynomial time. We proceed to show that this instance is equivalent to the MIS(k) instance. The first crucial step is given by the following lemma.

Lemma 7. *If $(\mathcal{T}, S, k') \in$ LSSCSEQ, then there exists a solution string $\widetilde{S}$ for $(\mathcal{T}, S, k')$ where $\widetilde{S}$ can be written as $\widetilde{S} := S'\, S^*\, T_1 -$ with*

$$S' := \widetilde{S}^1(c_1) \cdots \widetilde{S}^1(c_k) \,\&\, \widetilde{S}^2(c_1) \cdots \widetilde{S}^2(c_k) \,\&\, \widetilde{S}^3(c_1) \cdots \widetilde{S}^3(c_k),$$

such that for each $i \in \{1, \ldots, k\}$ we have:

– *$\widetilde{S}^1(c_i)$ is obtained from $S^1(c_i)$ by replacing exactly one occurrence of* 01 *to* 00*,*
– *$\widetilde{S}^2(c_i)$ is obtained from $S^2(c_i)$ by replacing exactly one occurrence of* 00 *to* 01*, and.*
– *$\widetilde{S}^3(c_i)$ is obtained from $S^3(c_i)$ by replacing exactly one occurrence of* 01 *to* 00*.*

Proof. Consider the suffix T_1- of $S-$. A careful examination shows that no matter what k changes are made to T_1-, the string S^* will not be a subsequence of the resulting string. This means that S^* is not a subsequence of the length $|T_1 - |$ suffix of $\widetilde{S}$, and so it must be a subsequence of the length $|S^*\, T_1 - |$ suffix of $\widetilde{S}$. Since S^* is contained as a substring in the $|S^*\, T_1 - |$ length suffix of $\widetilde{S}$, we can assume that $\widetilde{S}$ can be written as $\widetilde{S} := S'\, S^*\, T_1-$, and that S' contains as a subsequence the prefix $T_1 - S^*$ of T_1. Since for each $i \in \{1, \ldots, k\}$, this prefix contains three copies of each of $\overrightarrow{c_i}$ and $\overleftarrow{c_i}$, there must be three substrings $\widetilde{S}^1(c_i)$, $\widetilde{S}^2(c_i)$, and $\widetilde{S}^3(c_i)$ in S' that all begin with $\overrightarrow{c_i}$ and end with $\overleftarrow{c_i}$. Moreover, all of these $3k$ substrings must be disjoint in S'. It is now not difficult to see that the lemma follows due to our construction of the prefixes $T_1 - S^*$ and $T_2 - (S^* T_1 -)$ of our two enforcement strings $T_1, T_2 \in \mathcal{T}$. □

Let $\widetilde{S}$ be a solution string for $(\mathcal{T}, S, k')$ as in Lemma 7. We interpret the positions in S' that differ from $S-$ as a set of *selected vertices* $\{v_1^1, v_1^2, v_1^3, \ldots, v_k^1, v_k^2, v_k^3\}$ of G, where for each $i \in \{1, \ldots, k\}$, the vertex v_i^1 (resp. v_i^2, v_i^3) is the x-th vertex in c_i if the x-th substring 01 (resp. 00, 01) in $S(c_i)$ is modified in $\widetilde{S}(c_i)$. The next lemma shows that the set of selected vertices includes in fact only k vertices.

Lemma 8. *$v_i^1 = v_i^2 = v_i^3$ for each $i \in \{1, \ldots, k\}$.*

Proof. Consider some arbitrary $i \in \{1, \ldots, k\}$, and assume that v_i^1, v_i^2, and v_i^3 are the x-th, y-th, and z-th vertices colored c_i in G, respectively. Let $\widetilde{S}_i := \widetilde{S}^1(c_i) \,\&\, \widetilde{S}^2(c_i) \,\&\, \widetilde{S}^3(c_i)$, where $\widetilde{S}^1(c_i)$, $\widetilde{S}^2(c_i)$, and $\widetilde{S}^3(c_i)$ are the substrings of $\widetilde{S}$ given in Lemma 7. Since $\widetilde{S}_i$ contains the only occurrences of $\overrightarrow{c_i}$ and $\overleftarrow{c_i}$ in $\widetilde{S}$, it must contain as a subsequence any input string $T_e \in \mathcal{T}$, with e being an edge between two vertices colored c_i in G. Suppose that $x \neq y$. Consider the string $T(v_i^1) \,\&\, T(v_i^2) \in \mathcal{T}$. By construction, the only way this string can be a subsequence of $\widetilde{S}_i$ is if $y = z$. But then the string $T(v_i^2) \,\&\, T(v_i^1) \in \mathcal{T}$ is not a subsequence of $\widetilde{S}_i$. It follows that $x = y$. A similar argument can be used to show that $x = z$, and so $v_i^1 = v_i^2 = v_i^3$. □

According to Lemma 8, we let v_i be the single vertex corresponding to $v_i^1 = v_i^2 = v_i^3$, giving us a multicolored set $\{v_1, \ldots, v_k\}$ of vertices in G. The next lemma shows that this set must be independent in G.

Lemma 9. *The set of vertices $I := \{v_1, \ldots, v_k\}$ form an independent set in G.*

Proof. Suppose that there is an edge $e := \{v_i, v_j\}$ in G, $i < j$, and consider the substring $\widetilde{S}' := \widetilde{S}^1(c_i)\widetilde{S}^1(c_j) \,\&\, \widetilde{S}^2(c_i)\widetilde{S}^2(c_j) \,\&\, \widetilde{S}^3(c_i)\widetilde{S}^3(c_j)$. Since $\widetilde{S}'$ contains the only occurrences of $\overrightarrow{c_i}$, $\overleftarrow{c_i}$, $\overrightarrow{c_j}$ and $\overleftarrow{c_j}$ in $\widetilde{S}$, it must contain as a subsequence any input string $T_e \in \mathcal{T}$, with e being an edge between two vertices colored c_i and c_j in G. But by construction, the string $\widetilde{S}'$ does not contain as a subsequence the string $T_e := T(v_i) \,\&\, T(v_j) \in \mathcal{T}$, a contradiction. □

Combining Lemmas 7, 8, and 9 shows that if $(\mathcal{T}, S, k')$ has solution $\widetilde{S}$, then G has a multicolored independent set of size k. Conversely, starting at a multicolored independent set I in G, we can construct a string $\widetilde{S}$ corresponding to I as in Lemma 7 which can be easily verified to be a solution string for $(\mathcal{T}, S, k')$. Thus our reduction is correct, and by reducing the alphabet size as in Section 4, we obtain Theorem 4.

Corollary 3. LSSCSeq(k) *restricted to strings over a constant-size alphabet is* W[1]*-hard, and has no $n^{o(k)}$ algorithm unless ETH fails.*

6 Shortest Common Superstring

In this section we deal with a local search variant of the classical Shortest Common Superstring. In this problem, the input is a set of strings $\mathcal{T}$ and an integer ℓ, and the question is whether there is a string S of length at most ℓ which is a superstring of all strings in $\mathcal{T}$. The local search version of Shortest Common Superstring is defined as follows:

Local Search Shortest Common Superstring (LSSCStr):
Input: A set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of strings over an alphabet Σ, a string S which is a superstring of all T_i's, and a positive integer k.
Question: Is there a string $\widetilde{S}$ of length $|S| - 1$ which is a superstring of all T_i's such that $d_H(\widetilde{S}, S-) \leq k$?

Theorem 5. LSSCSTR(k) *is* W[1]*-hard, even with an alphabet of constant size.*

For ease of presentation, we describe here only the case that the alphabet size $|\Sigma|$ is part of the parameter. The case with constant-size alphabets can be coped with the method introduced in Section 4. The reduction is from the W[1]-complete MULTICOLORED CLIQUE problem, where, given a graph $G = (V, E)$ and a coloring function $c : V \to \{c_1, \ldots, c_k\}$, we ask for a multicolored clique of size k. We assume w.l.o.g. that c is a *proper* coloring, *i.e.* there is no edge $\{u, v\}$ between vertices u and v with $c(u) = c(v)$.

The basic idea is as follows. We construct an input string T_0, which is a suffix of the superstring S. After the removal of the last letter of S, T_0 has to be matched to a substring S' in S which encodes a vertex v from the first color class; this vertex represents the clique vertex selected from this color class. In order to match T_0, some modifications have to be performed in S'. These modifications then force k input strings, which are matched to substrings of S' before, to be matched to somewhere else in S. One of these input strings then forces modifications in another substring S'' of S, which corresponds to a vertex of the second color class. The other $k-1$ input strings encode pairs of color classes. Now, the modifications of S'' result in the selection of a vertex from the second color class, which then triggers a selection of a vertex of the third color class and so on.

After "selecting" one vertex from each color class, there are in total $k(k-1)$ input strings which encode color pairs, two strings for each color pair. If the selected vertices form a clique, then the two strings that correspond to the same pair can be matched to a substring of S which encodes the edge between the two selected vertices. In this case, one modification is sufficient for each pair of color classes; otherwise, two modifications are necessary. Hence, these $k(k-1)$ color pair strings can be matched with at most $k(k-1)/2$ modifications if and only if the selected vertices form a clique. The details are as follows.

The alphabet Σ consists of $k^* \cdot k(k-1)+4k+4$ letters with $k^* := 2^k(k^2+k)$. The letters \$ and # are separating letters, where \$ does not occur in the input strings. The letters 0 and 1 are encoding letters. The other letters correspond to colors and color pairs. For each color c_i, we have 4 letters: a_i, b_i, c_i, and d_i. For each ordered color pair (c_i, c_j) with $i \neq j$, there are k^* letters, namely, $c^1_{i,j}, \ldots, c^{k^*}_{i,j}$. Assume that each color class in G contains n vertices. The LSSCSTR(k)-instance consists of the superstring S and a set $\mathcal{T}$ of $1+(k-1)k \cdot n+(k-1) \cdot n$ input strings: one special input string T_0, k input strings for each vertex from color classes c_1 to c_{k-1}, and $k-1$ input strings for each vertex from the color class c_k.

To construct these strings, we first introduce some strings, which are used as "building blocks" in the construction. First, we describe the "separating blocks". For each color c_i with $2 \leq i \leq k$, we introduce two such blocks: $A_i := a_i^{g(i)}$ and $B_i := b_i^{g(i)}$, where $g(i) := 2^{k-i} \cdot (k^2+k)$. For each ordered pair of colors c_i and c_j with $i \neq j$, we construct one separating block: $C_{i,j} := (c^1_{i,j}\#)^n \cdots (c^{k^*}_{i,j}\#)^n$. Moreover, we construct two "color-pair matching" blocks for each color c_i:

- $M^1_i := 0C_{i,1} \cdots 0C_{i,i-1}\, C_{i,i+1} \cdots C_{i,k}$, and
- $M^2_i := C_{i,1} \cdots C_{i,i-1}\, C_{i,i+1}0 \cdots C_{i,k}0$.

Finally, for every vertex v we construct an "identifying block". Let $c_i := c(v)$. Here we distinguish $i = 1$ and $i > 1$. Assume v is the x'th vertex colored c_i. The identifying block for v is constructed as

- $I(v) := d_1\, 0^{x-1}\, 1\, 0^{n-x}\, d_1$ for $i = 1$, and
- $I(v) := d_i\, (0A_i)^{x-1} 1\, A_i (0A_i)^{n-x}\, d_i d_{i-1}$, for $i > 1$.

We are now ready to describe the set of input strings $\mathcal{T}$ in our LSSCSTR instance. First, for each vertex v colored c_i with $1 \leq i < k$, we construct one "triggering" input string. If v is the x'th

vertex colored c_i, its triggering input string $T(v)$ is constructed as:

$$T(v) := c_{i+1}\, M^1_{i+1}\, d_{i+1}\, (0A_{i+1})^{x-1} 0\, B_{i+1}\, (0A_{i+1})^{n-x}\, d_{i+1} d_i.$$

Then, for each vertex v colored c_i with $1 \leq i \leq k$, we add $k-1$ "pairing" input strings, each corresponding to a color class c_j with $j \neq i$. Here, we distinguish $i < j$ and $i > j$:

- $T(v, c_j) := C_{i,j+1} \cdots C_{i,k}\, I(v)\, C_{i,1} \cdots C_{i,i-1} C_{i,i+1} 0 \cdots C_{i,j} 0 \qquad (i < j)$,
- $T(v, c_j) := 0 C_{i,j} \cdots 0 C_{i,i-1} C_{i,i+1} \cdots C_{i,k}\, I(v)\, C_{i,1} \cdots C_{i,j-1} \qquad (i > j)$.

To finalize our construction of $\mathcal{T}$, we set the special input string T_0:

$$T_0 := c_1\, M^1_1\, d_1\, 0^n\, d_1.$$

Now, it remains to describe the temporary solution S. To this end, we introduce some further building blocks. For each edge $e = \{u, v\} \in E$, where u is colored c_i, and v is colored c_j with $i < j$, we construct one "edge block" $S(e)$ for S as:

$$S(e) := T(u, c_j) - 1 - T(v, c_i),$$

where $T(u, c_j)-$ as usual denotes the prefix of the pairing input string $T(u, c_j)$ without the last 0, and $-T(v, c_i)$ denotes the suffix of $T(v, c_i)$ without the first 0. Furthermore, for each vertex $v \in V$ colored c_i in G we construct the selection block $S(v)$ of v by:

- $S(v) := T(v)\, M^1_i\, I(v)\, M^2_i$ for $i < k$, and
- $S(v) := M^1_k\, I(v)\, M^2_k$ for $i = k$.

The solution S then consists of three parts, $S := S(V)S(E)T_0$, where the first part $S(V)$ is the concatenation of the selection blocks $S(v)$ separated by \$'s in any arbitrary order, the second part $S(E)$ is the concatenation of edge blocks $S(e)$ separated by \$'s, and T_0 is the special input string described above.

Finally, we set the parameter for the LSSCStr-instance to $k' := 2k + k(k-1)/2 + (2^{k-1} - 1)(k^2 + k)$. It is easy to verify that S is a superstring of all input strings: The string T_0 occurs at the end of S. Furthermore, for each vertex $v \in V$, the triggering input string $T(v)$ is a prefix of $S(v)$, while the pairing strings $T(v, c_j)$ are clearly substrings of $M^1_i\, I(v)\, M^2_i$. We next turn to showing the equivalence of the two instances.

Lemma 10. *If G has a multicolored clique K than $(S, \mathcal{T}, k')$ has a solution string $\widetilde{S}$.*

Proof. Let $K := \{v_1, \ldots, v_k\}$, where v_i is the vertex colored c_i in K, and assume v_i is the x_i'th vertex colored c_i in G. First, we change the only 1 in the identifying block $I(v_1)$ of the selection block $S(v_1)$ to 0 and the last character, *i.e.* d_1, of the triggering block $T(v_1)$ of $S(v_1)$ to c_1. Now T_0 is a substring of the resulting selection block $\widetilde{S}(v_1)$. Then, for every $1 < i \leq k$, we change the only 1 in the identifying block $I(v_i)$ in $S(v_i)$ to 0 and the x_{i-1}'th A_i-block in $I(v_i)$ is changed to a B_i-block. Moreover, the letter d_i at the end of the triggering block $T(v_i)$ is changed to c_i. Now, the triggering input string $T(v_{i-1})$ created for v_{i-1} is a substring of the resulting $\widetilde{S}(v_i)$. Next, consider the edges $e := \{v_i, v_j\}$, $i < j$, between the vertices in K. We change the only 1 in the edge block $S(e)$ in S to 0. After this, the pairing input strings $T(v_i, c_j)$ and $T(v_j, c_i)$ are substrings of the resulting $\widetilde{S}(e)$. After the modifications specified above, we get a string $\widetilde{S}$ of length $|S| - 1$ which is a superstring of all input strings. Summing up, we performed two modifications for v_1, $2 + |A_i|$ modifications for every $i > 1$, and one modification for each edge in K. This amounts to $k' = 2k + (2^{k-1} - 1)(k^2 + k) + k(k-1)/2$. □

We next consider the reversed direction. Suppose that a solution $\widetilde{S}$ exists for $(\mathcal{T}, S, k')$. We use $\widetilde{S}(v)$ to denote substring of $\widetilde{S}$ corresponding to the selection block $S(v)$ of S.

Lemma 11. *If there is a solution $\widetilde{S}$ for $(\mathcal{T}, S, k)$ constructed above, then the input string T_0 is a substring of some $\widetilde{S}(v_1)$ for some $v_1 \in V$ with $c(v_1) = c_1$.*

Proof. Since M_1^1 is a substring of T_0 and M_1^1 contains $(k-1) \cdot k^* > k'$ different letters $c_{1,j}^{\ell}$ with $1 \leq \ell \leq k^*$ and $2 \leq j \leq k$, T_0 cannot be a suffix of $\widetilde{S}$. For the same reason, T_0 cannot be a substring of $\widetilde{S}(v)$ for any vertex v colored c_i with $2 \leq i \leq k$. Moreover, since 0^n is a substring of T_0, T_0 cannot be substring of the edge blocks $\widetilde{S}(e)$ of $\widetilde{S}$. As \$ does not appear in T_0, this implies that T_0 has to be a substring of some $\widetilde{S}(v_1)$ with $c(v_1) = c_1$. □

Let v_1 be the vertex in Lemma 11. By construction, we have to match M_1^1 of T_0 to the M_1^1-substring of $\widetilde{S}(v_1)$. This implies that the letter 1 in the corresponding identifying block has to be changed to 0. Moreover, the last letter d_1 of the corresponding triggering block must be changed to c_1. These two changes cause that the pairing input strings and the triggering string for v_1 matched to somewhere else in $\widetilde{S}$ than in S. We consider first the triggering string $T(v_1)$.

Lemma 12. *The triggering input string $T(v_1)$ can only be matched a substring of $\widetilde{S}(v_2)$ for some vertex v_2 with $c(v_2) = c_2$.*

Proof. As in the proof of Lemma 11, the existence of M_2^1 in $T(v_1)$ prevents it to be matched to $\widetilde{S}(v)$ corresponding to a vertex colored c_i, $i > 2$, or to the edge blocks in $\widetilde{S}$. So the only possibilities are the triggering blocks $T(v)$ of $\widetilde{S}(v)$ with $c(v) = c_1$ and $v \neq v_1$, or $\widetilde{S}(u)$ for some vertex u colored c_2. Note that in the first case, we have one B_2-block in $T(v)$ and one B_2-block in $T(v_1)$, but they have different positions. This implies in order to match $T(v)$ to $T(v_1)$, we have to perform at least $2|B_2| = 2 \cdot 2^{k-2}(k^2 + k)$ modifications, more than the allowed number k'. □

As argued in the other proof direction, the matching of $T(v_1)$ to some $\widetilde{S}(v_2)$ causes $2 + |A_2|$ modifications after which the triggering input string $T(v_2)$ and $k-1$ pairing input strings are unmatched. With a similar proof as for Lemma 12, $T(v_2)$ has to be matched to some $\widetilde{S}(v_3)$ with $c(v_3) = c_3$: after performing the $2 + |A_2| = 2 + 2^{k-2}(k^2 + k)$ changes to match $T(v_1)$, one cannot afford to perform $2|B_3| = 2 \cdot 2^{k-3}(k^2 + k) = 2^{k-2}(k^2 + k)$ changes which are necessary for matching $T(v_2)$ to the selection block of another vertex colored c_2. The same argument applies inductively for all $i > 2$. In this way, the string $\widetilde{S}$ differs from S in exactly k selection blocks corresponding to a multicolored set of vertices $\{v_1, \ldots, v_k\}$ in G, and the Hamming distance between the remaining suffixes of $S-$ and $\widetilde{S}$ is at most $k(k-1)/2$.

Lemma 13. *The set of vertices $\{v_1, \ldots, v_k\}$ specified above form a clique in G.*

Proof. As stated above, to get T_0 and the $k-1$ triggering input strings matched with $\widetilde{S}$, we have to perform $(2^{k-1} - 1)(k^2 + k) + 2k$ modifications. Only $k(k-1)/2$ modifications remain for the $k(k-1)$ unmatched pairing strings. By construction, no pairing string is a substring of another pairing string. Consequently, the only possibility is to group them into pairs, where each pair has to form an edge, since then the pair can be matched to the corresponding edge block with only one modification, namely, changing the 1 to 0 in the edge block. This implies that the vertices whose selection blocks are changed from S to $\widetilde{S}$ must form a clique. □

Combining all lemmas above completes the proof of theorem in case $|\Sigma|$ is part of the parameter. A similar modification to the one done in Section 4 provides the proof for the constant-size alphabet

case. We mention that in the presented reduction, there are parts of the superstring S that are not matched to any input string, which is clearly not a feature of any minimal solution of Shortest Common Superstring. The reduction can be adapted such that this is not the case: First, replace each occurrence of \$ in S by one distinct string, for example, $\$1^i\$$ for the i'th occurrence, and add an input string equal to this string. By doing this, all letters in S occur in the input strings. Next, increase the length of T such that it contains all edge blocks as a prefix. Then, add a block in the beginning of S that contains some unique starting letter followed by the edge blocks. Finally, add one further input string that contains this unique starting letter and the edge blocks.

Corollary 4. *Unless ETH fails, there is no $n^{o(\lg k)}$ algorithm for* LSSCStr(k)*, even when all strings are over an alphabet of constant size.*

7 Conclusions

In this paper we addressed the question of whether the brute-force procedure for the local improvement step in a local search algorithm can substantially be improved when applied to four classical NP-hard string problems. In all cases we considered the Hamming distance as our metric for defining the local neighborhood where the search is performed, and showed that under this metric, the brute-force algorithm cannot be considerably improved for all four problems. There are, however, many other interesting metrics that deserve consideration. Furthermore, there are many other NP-hard string problems where local search can be applied. Finally, we mention that the local neighborhood defined for LSLCS and LSSCSub can also be defined by removing a letter at an *arbitrary* position from the old solution string S_{old} (and this is perhaps a more natural definition). However, our hardness results do not indicate a lower bound on the running-time of the local improvement step in this case; while we believe that this is the case, a proof of this is currently absent.

References

1. E. H. L. Aarts and J. K. Lenstra. *Local Search in Combinatorial Optimization*. Wiley-Interscience, 1997.
2. E. Balas. New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, 86:529–558, 1999.
3. J. Chen, B. Chor, M. Fellows, X. Huang, D. W. Juedes, I. A. Kanj, and G. Xia. Tight lower bounds for certain parameterized NP-hard problems. *Information and Computation*, 201(2):216–231, 2005.
4. M. Dörnfelder, J. Guo, C. Komusiewicz, and M. Weller. On the parameterized complexity of consensus clustering. In *Proc. 22nd ISAAC*, volume 7074 of *Lecture Notes in Computer Science*, pages 624–633. Springer, 2011.
5. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
6. M. R. Fellows, F. A. Rosamond, F. V. Fomin, D. Lokshtanov, S. Saurabh, and Y. Villanger. Local search: Is brute-force avoidable? In *Proc. 21st IJCAI*, pages 486–491, 2009.
7. P. Festa and P. M. Pardalos. Efficient solutions for the far from most string problem. *Annals of Operations Research*. To appear, available online.
8. F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Fast local search algorithm for weighted feedback arc set in tournaments. In *Proc. 24th AAAI*. AAAI Press, 2010.
9. J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37(1):25–42, 2003.
10. J. Guo, S. Hartung, R. Niedermeier, and O. Suchý. The parameterized complexity of local search for TSP, more refined. In *Proc. 22nd ISAAC*, volume 7074 of *Lecture Notes in Computer Science*, pages 614–623. Springer, 2011.
11. R. Impagliazzo and R. Paturi. Complexity of k-SAT. In *Proc. 14th CCC*, pages 237–240, 1999.
12. D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.
13. A. Krokhin and D. Marx. On the hardness of losing weight. *ACM Transactions on Algorithms*, 2012. To appear.

14. D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–71, 2011.
15. D. Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008.
16. D. Marx and I. Schlotter. Stable assignment with couples: Parameterized complexity and local search. *Discrete Optimization*, 8(1):25–40, 2011.
17. C. Meneses, C. A. S. Oliveira, and P. M. Pardalos. Optimization techniques for string selection and comparison problems in genomics. *IEEE Engineering in Medicine and Biology Magazine*, 24(3):81–87, 2005.
18. S. Szeider. The parameterized complexity of k-flip local search for SAT and MAX SAT. *Discrete Optimization*, 8(1):139–145, 2011.