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] Department of Industrial Management and Engineering,
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 be substantially improved when applied to classical NP-hard string problems. We examine four 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 is essentially optimal.

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 [11] and parameterized local search [13, 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 $f(k) \cdot \mathrm{poly}(n)$ time can be achieved. This is precisely the main question underlying the theory of *parameterized complexity* [5].

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 $4^k \cdot \mathrm{poly}(n)$ time. Marx [13] proved the non-existence of such an algorithm for the edge-exchange neighborhood. Subsequently, the complexity of local search for further neighborhood measures of Euclidean Traveling Salesman was examined [10]. Notably, the fixed-parameter tractability of Euclidean Travelling Salesman and the edge-exchange neighborhood remains open [13, 10]. Fellows et al. [6] provided fixed-parameter algorithms for local search variants of diverse graph problems such as 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 results concerning parameterized local search have been achieved for clustering problems [4], Boolean Constraint Satisfaction [12], Stable Marriage variants [14], and Satisfiability [16].

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 of the most 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 problems closely related to Closest String [7, 15].

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. We remark that these results do not exclude the existence of all parameterized local search algorithms for these problems, but rather motivate the study of further parameterizations, for instance by considering the combined parameter "number m of strings and neighborhood radius k".

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 analyze our local search string problems in the framework of parameterized complexity [5]. 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 $(x,k) \in \{0,1\}^* \times \mathbb{N}$ to an instance $(x',k') \in \{0,1\}^* \times \mathbb{N}$ such that:

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

If $g()$ is linearly bounded, *i.e.* $g(k) \leq ck$ for some constant c, then we say that the reduction is a *linear parameterized reduction*. Two basic classes of parameterized intractability are 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 L is W[1]-hard (W[2]-hard).

The hardness results in this paper are obtained by parameterized reductions from the following three problems which all have the solution size k as parameter: In the W[2]-hard Multicolored Hitting Set(k) (MHS(k)), the input is a hypergraph $(V, \mathcal{E})$ and a coloring function $c : V \to \{c_1, \ldots, c_k\}$. 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)), 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. We make use of the following result [3].[1]

Lemma 1. *Let L be a parameterized problem with parameter k, and assume that there is a linear parameterized reduction from either* MHS(k)*,* MIS(k)*, or* MC(k) *to L. Then unless all problems in* SNP *can be solved in subexponential time, size-n instances of L cannot be solved in $n^{o(k)}$ time.*

[1] For Hitting Set, Chen et al. [3] do not explicitly make this statement, but it can be inferred via a 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 $d^k \cdot poly(n, m)$ algorithm for this problem based on the following observation: as long as S differs from some input string at least d positions, then one of these positions in S has to be changed. Achieving an $f(m) \cdot \text{poly}(n)$-time algorithm by modifying the Integer Linear Programming-based algorithm of Gramm et al. [9] is also possible. Below, we show that for the parameter k, one cannot substantially improve on the brute-force algorithm in general, even when the strings are binary. We begin with the easier case of parameterized-size alphabets.

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

We next consider the binary case. 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 $T_\emptyset$.

Theorem 1. *There is a linear parameterized reduction from* MHS(k) *to* LSCS(k) *for binary strings.*

Corollary 1. LSCS(k) *for binary strings is* W[2]*-hard, it cannot be solved* $n^{o(k)}$ *time unless all problems in* SNP *can be solved in subexponential time.*

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 LCS:

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 $\binom{|S|}{k} \cdot |\Sigma|^k \cdot \text{poly}(n) = n^{O(k)}$ time by brute-force (n denotes the overall instance size). 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 2. *There is a linear parameterized reduction from the* W[2]*-hard* LCS(ℓ) *problem to* LSLCS(k) *with unbounded alphabets.*

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 $i \in \{1, \ldots, k\}$.

We begin by describing the solution string S. The string S consists of a suffix $S^* := (\$\pounds^{k+1}\$)^{k+1}$, where \$ and £ 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 $S(c_i) := \overrightarrow{c_i}(0\#)^n\overleftarrow{c_i}$ where $\overrightarrow{c_i}$ and $\overleftarrow{c_i}$ are letters corresponding to color class 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

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

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

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

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 $\overrightarrow{c_1}$, $\overleftarrow{c_1}$, $\ldots$, $\overrightarrow{c_k}$, $\overleftarrow{c_k}$, 0, 1, #, \$, and £. We now make two observations that lead to the soundness and completeness of our reduction.

Lemma 3. *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.*

According to Lemma 3 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 4. *The set $I \subseteq V(G)$ of vertices selected by $\widetilde{S}$ is a multicolored independent set in G.*

Theorem 2. *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 $i \in \{1,\dots,k\}$, 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 3 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 all problems in* SNP *can be solved in subexponential time.*

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 $\mathcal{T} = \{T_1,\dots,T_m\}$ 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 3. *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 $c : V \to \{c_1,\dots,c_k\}$. We assume, w.l.o.g., that there are n vertices colored c_i, for each color $c_i \in \{c_1,\dots,c_k\}$, 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.

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*:

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

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

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

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 $c_i \in \{c_1, \ldots, c_k\}$:

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

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 5. *Let $(\mathcal{T}, S, k')$ be an* LSSCSeq *instance constructed as described above. 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 *by* 00*,*
– $\widetilde{S}^2(c_i)$ *is obtained from* $S^2(c_i)$ *by replacing exactly one occurrence of* 00 *by* 01*,*
– $\widetilde{S}^3(c_i)$ *is obtained from* $S^3(c_i)$ *by replacing exactly one occurrence of* 01 *by* 00*.*

Let $\widetilde{S}$ be a solution string for $(\mathcal{T}, S, k')$ as in Lemma 5. 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 6. *For each* $i \in \{1, \ldots, k\}$ $v_i^1 = v_i^2 = v_i^3$*.*

According to Lemma 6, 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 is independent in G.

Lemma 7. *The set of vertices* $I := \{v_1, \ldots, v_k\}$ *forms an independent set in* G*.*

Corollary 3. LSSCSeq(k) *restricted to strings over a constant-size alphabet is* W[1]*-hard, and has no* $n^{o(k)}$ *algorithm unless all problems in* SNP *can be solved in subexponential time.*

6 Shortest Common Superstring

In this section we deal with a local search variant of 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 4. 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, that is, there is no edge $\{u, v\}$ between vertices u and v with $c(u) = c(v)$ (such edges can be removed in linear time), and that each color class contains exactly $|V|/k$ vertices.

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, $c^1_{i,j}, \dots, c^{k^*}_{i,j}$. 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 $i > j$:

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

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:

- $S(v) := T(v)\, M_i^1\, I(v)\, M_i^2$ for $i < k$, and
- $S(v) := M_k^1\, I(v)\, M_k^2$ 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 $M_i^1\, I(v)\, M_i^2$. We next turn to showing the equivalence of the two instances.

Lemma 8. *If G has a multicolored clique K then $(S, \mathcal{T}, k')$ has a solution string $\widetilde{S}$.*

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 9. *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 $c(v_1) = c_1$.*

Let v_1 be the vertex in Lemma 9. 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 are matched to somewhere else in $\widetilde{S}$ than in S. We consider first the triggering string $T(v_1)$.

Lemma 10. *The triggering input string $T(v_1)$ can only be matched to a substring of $\widetilde{S}(v_2)$ for some vertex v_2 with $c(v_2) = c_2$.*

It can be shown that 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. Furthermore, $T(v_2)$ 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 11. *The set of vertices $\{v_1, \ldots, v_k\}$ specified above forms a clique in G.*

Combining all lemmas above completes the proof of Theorem 4 when $|\Sigma|$ is part of the parameter. Using the method from Section 4 gives the proof for constant-size alphabets.

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. *Ann. Oper. Res.*, 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. *Inform. Comput.*, 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 *LNCS*, 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? *J. Comput. Syst. Sci.*, 78(3):707–719, 2012.
7. P. Festa and P. M. Pardalos. Efficient solutions for the far from most string problem. *Ann. Oper. Res.*, 196(1):663–682, 2012.
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 *LNCS*, pages 614–623. Springer, 2011.
11. D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.
12. A. Krokhin and D. Marx. On the hardness of losing weight. *ACM T. Alg.*, 8(2):19, 2012.
13. D. Marx. Searching the k-change neighborhood for TSP is W[1]-hard. *Oper. Res. Lett.*, 36(1):31–36, 2008.
14. D. Marx and I. Schlotter. Stable assignment with couples: Parameterized complexity and local search. *Discr. Optim.*, 8(1):25–40, 2011.
15. C. Meneses, C. A. S. Oliveira, and P. M. Pardalos. Optimization techniques for string selection and comparison problems in genomics. *IEEE Eng. Med. Bio. Mag.*, 24(3):81–87, 2005.
16. S. Szeider. The parameterized complexity of k-flip local search for SAT and MAX SAT. *Discr. Optim.*, 8(1):139–145, 2011.