On Self-Reducibility and Reoptimization of Closest Substring Problem

Jeffrey A. Aborot, Henry Adorna, Jhoirene B. Clemente
Algorithms and Complexity Laboratory
Department of Computer Science
College of Engineering
University of the Philippines Diliman
jeffrey.aborot@up.edu.ph, ha@dcs.upd.edu.ph, jbclemente@up.edu.ph

ABSTRACT

In this paper, we define the reoptimization variant of the closest substring problem (CSP) under sequence addition. We show that, even with the additional information we have about the problem instance, the problem of finding a closest substring is still NP-hard. We investigate the combinatorial property of optimization problems called self-reducibility. We show that problems that are polynomial-time reducible to self-reducible problems also exhibits the same property. We illustrate this in the context of CSP. We used the property to show that although we cannot improve the approximability of the problem, we can improve the running time of the existing PTAS for CSP.

Keywords
reoptimization, closest substring problem, approximation

1. INTRODUCTION

The *consensus pattern problem* is a combinatorial problem applied to a wide range of applications from string matching problems in genomic sequences [4] to finding repeated patterns in graphs [7] and time-series databases [11]. Due to its utmost importance in the field of genomics, several efforts have been made to characterize the computational requirements needed to solve the problem. If the set of input instances is constrained only to a set of input strings, the problem is known as the *closest substring problem* (CSP). The problem seeks to identify a pattern that occurs approximately in each of the given set of sequences.

It has been shown that the CSP is NP-hard [9], i.e., unless P=NP, we cannot obtain a polynomial-time algorithm solving the problem. Moreover, it was also shown that fixing parameters such as pattern length or alphabet size, does not address the intractability of the problem [8]. Aside from parameterization attempts, other studies tried to relax the condition of always finding the optimal solution by

providing approximation algorithms for the problem. The first constant-factor approximation algorithm is presented in [10], then subsequently improved in [12]. These results include CSP to the class of problems that are constant-factor approximable (APX). In addition to this, several studies [12, 13, 14] even presented a *polynomial-time approximation scheme* (PTAS) for the problem.

To address the intractability and to improve the solution quality of approximation algorithms, another approach is to use additional information about the problem instance whenever possible. A method called *reoptimization* has already been applied to a variety of hard problems in the literature. The idea of reoptimization is to make use of a solution to a locally modified version of the input instance. It was shown that reoptimization can help to either improve the approximability and even provide a PTAS for some problems that are APX-hard. These results include improvements for the metric-traveling salesman problem [3], the Steiner tree problem [2], the common superstring problem [1], and hereditary graph problems [5].

In this paper, we investigate whether reoptimization can help in approximating the closest substring problem. The additional information given in advance is a solution to a smaller instance of the problem. The algorithm for the reoptimization variant of the problem aims to make use of the given solution to become feasible for the larger instance. We show that, using the given solution as a greedy partial solution to the larger instance, we can achieve an additive error (with respect to the optimal solution) that grows linearly as we increase the number of sequences added to the original instance, which can be worse than the existing σ-approximation algorithms for the original problem.

The self-reducibility property of some hard combinatorial problem has been showed to improve the approximability of the problem. However, existing general approaches incur a much longer time for providing a solution with improved quality. Providing an improved ratio is already possible for CSP, due to the existence of a PTAS. However, [17] deemed the PTASs in the literature as impractical for small error bounds. In this paper, we showed that it is possible to improve the running time of the existing PTAS for CSP while maintaining the same approximation ratio through reoptimization.

2. CLOSEST SUBSTRING PROBLEM

Given a set of sequences $\mathcal{S} = \{S_1, S_2, \ldots, S_t\}$ defined over some alphabet Σ, where each S_i is of length n, and for some $l \leq n$, find a string v of length l and a set of substrings y_i each from S_i, where $v, y_i \in \Sigma^l$ such that the total Hamming distance $\sum_i^t d(v, y_i)$ is minimized [9].

The closest substring of a given set of input sequences, may not be unique given the closest substring occurrences y_i, for $1 \leq i \leq t$. So instead of focusing on the closest substring v, we will use the collection of y_i's to represent a feasible solution for the problem. We can represent a feasible solution

$$SOL = (y_1, y_2, \ldots, y_t)$$

using a sequence of substrings obtained from each of the given sequences to $\mathcal{S}$. We may refer to SOL as a feasible solution and OPT as the optimal solution for input instance $\mathcal{S}$.

We can easily obtain a substring v from the collection of occurrences SOL. This is done by aligning all substrings in SOL and taking the majority symbol for each column of the alignment. Let us call this string the *consensus* of SOL. Moreover, the consensus of OPT is called the closest substring of $\mathcal{S}$. It is easy to see that the consensus of OPT will minimize the total Hamming distance for all $v \in \Sigma^l$. On the other hand, given a string $v \in \Sigma^l$, we can also obtain a set of occurrences SOL by searching the best aligning substrings from each of the sequences in $\mathcal{S}$. Given the closest substring, we can obtain OPT in $O(tn)$.

Naively, we can search for the optimal solution by considering all possible locations of the closest substring in each sequence S_i from $\mathcal{S}$. For smaller alphabet sizes, it is easier to search over all Σ^l as compared to $O(n^t)$ possible occurrences. However, since we are considering the general CSP, we do not restrict the alphabet size of the given sequences.

2.1 Related Works

Algorithms for the CSP can be categorized into three classes: *exact*, *heuristics*, and *approximate*. Exact algorithms always obtain the optimal solution, but are exhaustive and impractical due to the NP-hardness of the problem. Among the earliest exact algorithms, some use graphs to model the problem. This includes WINNOWER [15] which involves finding cliques in graphs obtained from the set of sequences. An integer linear programming (ILP) formulation of the problem was also presented in [18].

Some algorithms make use of some strategy in searching through the set of all feasible solutions. Algorithms following this approach are called *heuristic algorithms*. A majority of these results use probabilistic models to represent solutions. When it is possible to guarantee the quality of the solution by means of identifying the bounds of its solution's cost, it is called an *approximation algorithm*. Since we do not know the cost of an optimal solution, certain properties of the input instances and the problem itself are needed to design the algorithm. For the CSP, the first constant-factor 2-approximation algorithm was presented in [10], which was subsequently improved in [12].

PTAS	Approximation Ratio	Running Time
[Li1999]	$1 + \frac{4|\Sigma|-4}{\sqrt{e}(\sqrt{4r+1}-3)}$	$O(l(tn)^{r+1})$
[Li1999]	$1 + O(\sqrt{\frac{\log r}{r}})$	$O(l(tn)^{r+1})$
[Ma2000]	$1 + \frac{1}{2r-1} + 3\epsilon r$	$O(l(tn)^{r+1}|\Sigma|^{\sqrt{4/\epsilon^2}\log tn})^1$

Table 1: Summary of approximation ratio and running time of the existing PTASs for CSP in the literature.

Due to the hardness results presented in [4], several other efforts were made to identify for which types of input instances the problem becomes easier to solve. A result in [6] shows that even if the set of input instances is defined over the binary alphabet, we still cannot obtain a practical polynomial-time algorithm for small error bounds. Aside from characterizing input instances, one line of research focused on the *parameterization* of the problem [8]. Based on these studies, it is shown that the problem is *fixed-parameter intractable*, i.e., fixing a parameter such as the pattern length or alphabet size will not make the problem easier to solve.

A PTAS for a hard problem is set of polynomial-time algorithms for which one can specify a certain guaranteed quality. However, since there is a trade-off between the quality of solution and the running time of the algorithm, we can expect a longer running time for smaller error bounds.

The summary of the approximation ratio and running time of the existing PTASs for CSP is shown in Table 1. The second on the list is a randomized PTAS for CSP, while the third one assumes a general alphabet Σ. We can see that the approximation ratio of the third PTAS is only dependent on the sampling size r, because the alphabet size is compensated in the running time of the algorithm. In contrast, we have the alphabet size as a parameter in the approximation ratio of the first PTAS.

We consider the first PTAS in Table 1 for our study, since we will only focus on deterministic algorithms for reoptimization. Moreover, we assume that the set of input instances are obtained from a general alphabet Σ and so choosing the third PTAS may not be ideal in our case since it will not just have r as a parameter but also the alphabet size.

The first PTAS from [12] is shown in Algorithm 1. For each parameter r, it describes an approximation algorithm for CSP that outputs a solution SOL with

$$cost(SOL) \leq \left(1 + \frac{4|\Sigma|-4}{\sqrt{e}\sqrt{4r+1}-3}\right) \cdot cost(OPT)$$

in $O(l(tn)^{r+1})$ time.

An *r-sample* from a given instance $\mathcal{S}$, i.e.,

$$r\text{-}sample(\mathcal{S}) = \{y_{i_1}, y_{i_2}, \ldots, y_{i_r}\},$$

is a collection of r l-length substrings from $\mathcal{S}$. Repetition of substrings are allowed for as long as no two substrings are obtained from the same sequence. Let $R(\mathcal{S})$ denote the set of all possible *r-sample* from $\mathcal{S}$. The total number of samples

in $\mathcal{S}$ is $\binom{tn}{r}$ which is bounded above by $O((tn)^r)$. From each sample, the algorithm obtains a consensus pattern. Solution SOL is then derived by aligning the t closest substrings from the given consensus. The Algorithm 1 minimizes through all possible r-*sample* in $\mathcal{S}$ to provide a feasible solution with a guaranteed quality.

Algorithm 1 PTAS for the CSP [Li1999]

Input: Set of sequences $\mathcal{S}$, pattern length l, sampling size r

Output: SOL and consensus string v_{sol}

1: $min = \infty$
2: **for each** r-*sample* $\{y_{i_1}, \ldots, y_{i_r}\}$ from $\mathcal{S}$ **do**
3:     $v = $ consensus pattern from $\{y_{i_1}, \ldots, y_{i_r}\}$
4:     $SOL = \emptyset$
5:     **for each** $S_i \in \mathcal{S}$ **do**
6:         $y_i = \min\limits_{\forall y_i \in S_i} d(v, y_i)$
7:         $SOL = $ merge y_i to SOL
8:     **end for**
9:     **return** SOL with $\min cost(SOL)$
10: **end for**

---

3. SELF-REDUCIBILITY

A solution to a combinatorial optimization problem is composed of a set of discrete elements called *atoms* [19]. For certain graph problems, it can be the set of all vertices or set of all edges, e.g., a clique in the maximum clique problem is a collection of vertices.

A problem Π is said to be *self-reducible* if there is a polynomial-time algorithm, Δ, satisfying the following conditions [19].

1. Given an instance I and an atom α of a solution to I, Δ outputs an instance I_α. We require that the size of I_α is smaller than the size of I, i.e., $|I_\alpha| < |I|$. Let $\mathcal{R}(I|\alpha)$ represent the set of feasible solutions to I containing the atom α. We require that every solution SOL of I_α, i.e., $SOL \in \mathcal{R}(I_\alpha)$, has a corresponding $SOL \cup \{\alpha\} \in \mathcal{R}(I|\alpha)$ and that this correspondence is one-to-one.

2. For any set $H \in \mathcal{R}(I_\alpha)$ it holds that the
$$cost(I, H \cup \{\alpha\}) = cost(I_\alpha, H) + cost(I, \alpha).$$

Given the properties of a self-reducible problem, we prove that the following lemma is true.

LEMMA 1. *CSP is self-reducible.*

PROOF. With the assumption that the pattern length l is constant, a valid input instance I to CSP is a set of t sequences $\mathcal{S}$. A feasible solution $SOL \in \mathcal{R}(\mathcal{S})$ is an ordered set $SOL = (y_1, y_2, \ldots, y_t)$. The set of atoms in CSP are all possible l-length substrings in $\mathcal{S}$. Let us define a reduction function $\Delta(\mathcal{S}, y_i)$, which returns a reduced instance $\mathcal{S} \setminus \{S_i\}$. The reduced instance is derived by removing one sequence S_i where y_i is obtained. We argue next that $\Delta(\mathcal{S}, y_i)$ has the following properties.

1. For a $SOL \in \mathcal{R}(\mathcal{S})$ there is a corresponding $SOL_\alpha \cup \{\alpha\} \in \mathcal{R}(I|\alpha)$. Note that, $SOL = (y_1, y_2, \ldots, y_t)$ guarantees at least one occurrence of the closest substring per sequence. For any atom y_i, a solution $SOL \in \mathcal{R}(\mathcal{S}|y_i)$ corresponds to $SOL_i \oplus y_i$, i.e.,
$$(y_1, y_2, \ldots, y_{i-1}, y_{i+1}, \ldots, y_t) \oplus y_i \in \mathcal{R}(\mathcal{S}|y_i).$$
Instead of using the operation '$\cup$' for sets, we used $\oplus$ to denote the merge of an element to a sequence.

2. For any feasible solution SOL_i to the reduced instance $\mathcal{S} \setminus \{S_i\}$, we can obtain a feasible solution $SOL_i \oplus y_i$ for $\mathcal{S}$ with $cost(\mathcal{S}, SOL_i \oplus y_i)$ equal to the sum of $cost(\mathcal{S} \setminus \{S_i\}, SOL_i)$ and $cost(\mathcal{S}, y_i)$, since
$$\begin{aligned} cost(SOL_i \oplus y_i) &= cost(\mathcal{S} \setminus \{S_i\}, SOL_i) + cost(\mathcal{S}, y_i) \\ &= \sum_{\substack{i=1 \\ j \neq i}}^{t} d(v, y_j) + d(v, y_i), \end{aligned}$$
where v is the consensus of $SOL_i \oplus y_i$.

$\square$

Let us use the concept of polynomial-time reduction for combinatorial problems. We say that a problem A is polynomial-time reducible to B, denoted by $A \leq_P B$, if $\exists$ a polynomial-time transformation f, which for every input
$$x \in A \leftrightarrow f(x) \in B.$$

In other words, in order to solve problem A, we must at least solve problem B. The problem of finding the closest substring is reduced to a graph problem in [15]. The transformation is as follows. Given a set of sequences $\mathcal{S}$ and a pattern length l, an edge weighted t-partite graph $G_\mathcal{S} = (V, E, c)$ is obtained, where the problem is reduced to finding a minimum weighted clique on a t-partite graph (MWCP). Each substring in $\mathcal{S}$ represents a vertex in $G_\mathcal{S}$. A part $V_i \subset V$ represents the set of vertices obtained from a single sequence $S_i \in \mathcal{S}$. A vertex $v \in V_i$ is connected to all other vertices in V except those belonging to V_i. The cost defined by function $c : (V \times V) \to \mathbb{Z}^+$ between two vertices can be interpreted as the Hamming distance between two substrings in $\mathcal{S}$. The cost of a clique in $G_\mathcal{S}$, which is computed by getting the sum of all the edges is equal to sum of all pairwise Hamming distances of substrings in SOL in $\mathcal{S}$. Moreover, we can create an instance $G_\mathcal{S}$ from $\mathcal{S}$ in polynomial-time. Thus, showing a polynomial-time reduction from CSP to MWCP.

It is also shown that MWCP has an exact reduction to Minimum Weighted Independent Set Problem (MWISP), since a clique in a graph is an independent set in the corresponding complement of the graph [9]. In line with this we would like to cite the following Lemma from [19].

LEMMA 2. *Maximum Weighted Independent Set Problem (MWISP) is self-reducible.*

Since we proved earlier that CSP is self-reducible and we know that there is a polynomial-time reduction from CSP

to MWCP, which is equivalent to a self-reducible problem MWISP, we are interested to know if all problems that are polynomial-time reducible to self-reducible problems also exhibit the same property.

THEOREM 1. *If problem A is polynomial-time reducible to problem B ($A \leq_P B$), and B is self-reducible, then A is self-reducible.*

PROOF. Let $A = (\mathcal{D}_A, \mathcal{R}_A, cost_A, goal_A)$ and $B = (\mathcal{D}_B, \mathcal{R}_B, cost_B, goal_B)$ be two NP optimization problems. Let $I_A \in \mathcal{D}_A$, $SOL_A \in \mathcal{R}(I_A)$, where SOL_A is composed of atoms α_A. Similarly, let $I_B \in \mathcal{D}_B$, $SOL_B \in \mathcal{R}(I_B)$, where SOL_B is composed of atoms α_B.

Given that $A \leq_P B$, then by definition, there exists a polynomial-time computable function f such that for every instance I_A for problem A, $f(I_A)$ is an instance of problem B and for every solution SOL_A to A, $f(SOL_A) \in \mathcal{I}_B$. Equivalently, as a decision problem, the polynomial-time reducibility implies

$$(I_A, SOL_A) \in (\mathcal{D}_A \times \mathcal{R}(\mathcal{I}_A))$$

$$\leftrightarrow$$

$$(f(I_A), f(SOL_A)) \in (\mathcal{D}_B \times \mathcal{R}(\mathcal{I}_B))$$

By definition of self-reducibility, if B is self-reducible then we can obtain a self-reduction function $\Delta_B(I_B, \alpha_B) = I_{\alpha_B}$, such that $|I_{\alpha_B}| < |I_B|$ and the following conditions hold.

1. For $SOL_B \in \mathcal{R}(I_B)$ there is a corresponding $SOL_{\alpha_B} \cup \{\alpha_B\} \in \mathcal{R}(I_B|\alpha_B)$, where $SOL_{\alpha_B} \in \mathcal{R}(I_{\alpha_B})$.

2. For a subset of atoms $H_B \subseteq \mathcal{R}(I_{\alpha_B})$, $cost_B(I_B, H_B \cup \{\alpha_B\}) = cost_B(I_B, \alpha_B) + cost_B(I_{\alpha_B}, H_B)$.

Given that B is self-reducible, we need to show that problem A is also self-reducible. If so, we must construct a self-reduction function $\Delta_A(I_A, \alpha_A) = I_{\alpha_A}$ that follows the two properties.

Given the polynomial-time function f and the self-reduction function Δ_B, we realize Δ_A using Δ_B through the following,

$$\Delta_B(f(I_A), f(\alpha_A)) = f(I_{\alpha_A}).$$

The self-reduction function for Δ_A clearly inherits the two conditions because of our premise that B is self-reducible. Moreover, the reduction function runs in polynomial-time since f is polynomial-time computable. $\square$

Theorem 1 is presented in the hope that we can use the current approaches for the reoptimization variants of clique and independent set problem for providing improvements for CSP. It is shown that for some defined reoptimization variant of clique and independent set, a general method is shown to improve the approximability with trade-off on the running time of the approximation algorithm.

4. REOPTIMIZATION

For real-world applications, additional information about the problems we are solving often is available and so we may not have to solve them from scratch. One of the approaches is to make use of a priori information, which can be a solution to a smaller input instance of a problem to solve a larger instance of it. This approach is called *reoptimization*. The idea was first mentioned in [16]. For some problems, we can transform the given optimal solution so that it may become feasible for the modified instance in polynomial-time. Furthermore, this approach can help to improve the approximability of the problem or the running time of the algorithms solving it. In fact, we can obtain a PTAS for a reoptimization variant of some problem given that the unmodified problem has a constant-factor approximation algorithm [3].

In the reoptimization variant of any problem, it is important to define precisely the modification relation $\mathcal{M}$ over the set of input instances. For graph problems, common modifications involve addition/deletion of edges and vertices. Other types of modification involve changes in the edge/vertex weights. A simple definition of reoptimization is as follows.

INPUT: Original instance I, its optimal solution OPT, and a modified instance I', where $(I, I') \in \mathcal{M}$
OUTPUT: Solution SOL' to I'.

For the CSP, we consider the basic type of modification where one or several sequences are added to $\mathcal{S}$. Since the length of the pattern remains unchanged, we will only define the modification relation over the given set of sequences. When a single sequence is added to $\mathcal{S}$, we have the modification relation $\mathcal{M}_+$, where $(\mathcal{S}, \mathcal{S}') \in \mathcal{M}_+$, if $\mathcal{S}' = \mathcal{S} \cup \{S_{t+1}\}$ and $S_{t+1} \notin \mathcal{S}$. As a generalization, we define the modification relation $\mathcal{M}_{k+}$ to denote addition of k sequences to $\mathcal{S}$, i.e., $(\mathcal{S}, \mathcal{S}') \in \mathcal{M}_{k+}$, if $\mathcal{S}' = \mathcal{S} \cup \{S_{t+1}, S_{t+2}, \ldots, S_{t+k}\}$ and $S_{t+i} \notin \mathcal{S}$, for $1 \leq i \leq k$.

Let us define the reoptimization variant of the CSP under single sequence addition. In the following definition, we can see that we already have the optimal solution with closest substring pattern v_{opt}, of the original instance $\mathcal{S}$. Note that we can easily compute occurrences of v_{opt} in $\mathcal{S}$, i.e., the set of substrings y_i, each from S_i and their positions where v_{opt} is obtained.

DEFINITION 1. *Reopt-CSP$_{\mathcal{M}_+}$*
INPUT: *Pattern length l, original instance $\mathcal{S}$, the optimal closest substring v_{opt} of $\mathcal{S}$, and a modified instance $\mathcal{S}'$ where $(\mathcal{S}, \mathcal{S}') \in \mathcal{M}_+$*
OUTPUT: *Solution v'_{sol} to the modified instance $\mathcal{S}'$.*

Even with the additional information we have, the reoptimization variant of the problem is still NP-hard, i.e., no polynomial-time algorithm exists to obtain the optimal solution for $\mathcal{S}'$, unless $P = NP$.

THEOREM 2. *Reopt-CSP$_{\mathcal{M}_+}$ is NP-hard.*

PROOF. Towards contradiction, assume that $Reopt\text{-}CSP_{\mathcal{M}_+}$ is polynomial-time solvable. We will make use of the polynomial-time algorithm for $Reopt\text{-}CSP_{\mathcal{M}_+}$ to solve the CSP.

We start by showing that for two sequences, both of length n, in $\mathcal{S}$, we can obtain the optimal closest substring v_{opt} in polynomial-time. We can obtain v_{opt} in $O(l \cdot n^2)$ by exhausting all possible l-length patterns in both sequences.

By making use of the optimal solution v_{opt} and a polynomial-time algorithm for $Reopt\text{-}CSP_{\mathcal{M}_+}$, we can solve the CSP for any number of sequences in $\mathcal{S}$ in polynomial-time by adding one sequence at a time. But then, we know that closest substring is NP-hard. Therefore, $Reopt\text{-}CSP_{\mathcal{M}_+}$ is also NP-hard.

$\square$

4.1 Approximation Algorithms for CSP

For the purpose of our discussion, we may refer to OPT and OPT' to be the optimal solution of the smaller instance and the larger instance of the problem, respectively. The output of the presented approximation algorithms is denoted by SOL', unless otherwise stated.

First, we show how we can use a simple algorithm to give an approximate solution for $Reopt\text{-}CSP_{\mathcal{M}_+}$. The algorithm makes use of the given solution as a greedy partial solution to the larger instance.

Algorithm 2 Given a sequence $S_{t+1} \in \Sigma^n$ and solution SOL of input instance $\mathcal{S}$, procedure BEST-ALIGN produces a feasible solution SOL' by aligning the closest substring from S_{t+1} to v_{sol}

1: **procedure** BEST-ALIGN(SOL, S_{t+1})
2:     v_{sol} = Consensus pattern from SOL
3:     $min = \infty$
4:     **for each** l-length substring x of S_{t+1} **do**
5:         **if** $d(v_{sol}, x) < min$ **then**
6:             $min = cost(x)$
7:             $y_{t+1} = x$
8:         **end if**
9:     **end for**
10:     $SOL' = (y_1, \ldots, y_t, y_{t+1})$
11:     **return** SOL'
12: **end procedure**

Algorithm 2 searches for the best aligning substring y_{t+1} to the given solution. In adding one sequence to $\mathcal{S}$, the computed solution may or may not be the optimal solution for the larger instance.

For the first case, if OPT is subset of OPT' for $\mathcal{S}'$, then the result of Algorithm 2 will yield the optimal solution for $\mathcal{S}'$. Otherwise, there exists a non optimal solution for $\mathcal{S}$ that is part of the optimal solution for $\mathcal{S}'$, i.e., $\exists SOL \subset OPT'$. The solution SOL' of Algorithm 2 is obtained by merging the given optimal solution OPT with the best possible aligning substring in the new sequence. Algorithm 2 obviously runs in linear time with respect to the length of the additional sequence S_{t+1}. To illustrate that we cannot always

get the optimal solution using Algorithm 2, let us consider the following example.

EXAMPLE 1. *Let $\mathcal{S} = \{S_1, S_2, S_3, S_4\}$ with S_5 as the additional sequence for the modified input $\mathcal{S}'$. If we are looking for a closest substring of length $l = 4$, the optimal solution for $\mathcal{S}$ is $v_{opt} : AAAA$.*

$S_1:$	**A**	**A**	**A**	**A**	B	B	B	B
$S_2:$	B	B	B	B	**A**	**A**	**A**	**A**
$S_3:$	**A**	**A**	**A**	**A**	B	B	B	A
$S_4:$	B	B	B	B	**A**	**A**	**A**	**A**
$S_5:$	**B**	**B**	**B**	**B**	B	B	B	B

Algorithm 2 will return a solution by aligning the most similar substring $y_5 : BBBB$ from the new sequence S_5. The solution produced by Algorithm 2 will have $cost(SOL') = 4$. On the other hand, the optimal solution for $\mathcal{S}'$ is $v'_{opt} : BBBB$ with $cost(OPT') = 1$.

In this case, the subset of the optimal solution for the larger instance is not the optimal solution for the smaller instance. Algorithm 2 obviously runs in linear time with respect to the length of the additional sequence S_{t+1}, multiplied by getting the cost of each substring x.

To get the quality of the approximation algorithm, we need to compare $cost(OPT')$ and $cost(SOL')$. Using Algorithm 2 and the second property of self-reducibility, we have

$$cost(SOL') = cost(OPT) + d(v_{opt}, y_{t+1}),$$

where y_{t+1} is the best aligning substring to v_{opt} from the new sequence. We know that the cost of the optimal solution for the smaller instance is less than or equal to the cost of the optimal solution for the larger instance, i.e., $cost(OPT) \leq cost(OPT')$. Therefore,

$$cost(SOL') \leq cost(OPT') + d(v_{opt}, y_{t+1}).$$

In the worst case scenario, we can get

$$cost(SOL') \leq cost(OPT') + l.$$

Since we only added a single sequence, the quality of the solution depends solely on how long the pattern is. Using Algorithm 2, we can get a feasible solution for $\mathcal{S}'$ in $O(ln)$. Compared to the 2-approximation algorithm for CSP with running time $O(l(tn)^2)$ from [10], our approach can benefit an improved approximation ratio and running time for instances $\mathcal{S}'$ where $OPT' < l$. Note that, the first property of the self-reducibility of CSP allows us to provide a feasible solution for the modified instance by extending the additional information that we have. We illustrate in the following generalization that it is possible to produce a feasible solution even if we add k-sequences to $\mathcal{S}$.

4.2 Generalization

In this section, we consider the extension of the reoptimization variant where, instead of adding one sequence, the modification relation, denoted by $\mathcal{M}_k^+$, is characterized by

adding k sequences to the original instance. The definition of the generalized reoptimization version under sequence addition is as follows.

DEFINITION 2. $Reopt\text{-}CSP_{\mathcal{M}_k^+}$
INPUT: Pattern length l, original instance $\mathcal{S}$, the optimal closest substring v_{opt} of $\mathcal{S}$, and a modified instance $\mathcal{S}'$ where $(\mathcal{S}, \mathcal{S}') \in \mathcal{M}_k^+$
OUTPUT: Solution v'_{sol} to the modified instance $\mathcal{S}'$.

Since $Reopt\text{-}CSP_{\mathcal{M}_k^+}$ is a generalization of $Reopt\text{-}CSP_{\mathcal{M}_+}$, where $k = 1$, we no longer need to show that this variant is also NP-hard. To give an approximate solution, we can generalize Algorithm 2 for $Reopt\text{-}CSP_{\mathcal{M}_k^+}$.

Algorithm 3 Given k additional sequences $\{S_{t+1}, \ldots S_{t+k}\}$, where each $S_{t+i} \in \Sigma^n$ and solution SOL for input instance $\mathcal{S}$, procedure K-BEST-ALIGN produces a feasible solution SOL' by aligning the closest substrings from each of the additional sequences to v_{sol}

1: **procedure** K-BEST-ALIGN($SOL, \{S_{t+1}, \ldots S_{t+k}\}$)
2:    $v_{sol} =$ Consensus pattern from SOL
3:    **for each** i in 1 to k **do**
4:        $min = \infty$
5:        **for each** l-length substring x of S_{t+i} **do**
6:            **if** $d(v_{sol}, x) < min$ **then**
7:                $min = cost(x)$
8:                $y_{t+i} = x$
9:            **end if**
10:        **end for**
11:    **end for**
12:    $SOL' = (y_1, \ldots, y_t, y_{t+1}, \ldots, y_{t+k})$
13:    **return** SOL'
14: **end procedure**

Given v_{opt}, we can give a feasible solution for $\mathcal{S}'$ by getting the best aligning substring y_{t+i} from each of the newly added sequences. Let Y be the set of substrings y_{t+i} and let

$$cost(Y) = \sum_{i=1}^{k} d(v_{opt}, y_{t+i})$$

be the contribution of the set Y to SOL'. If v_{opt} is the same as v'_{opt}, then we can always guarantee optimality, otherwise

$$cost(SOL') \leq cost(OPT') + cost(Y).$$

In the worst case scenario, we can have

$$cost(SOL') \leq cost(OPT') + kl.$$

The extension of Algorithm 3 for $Reopt\text{-}CSP_{\mathcal{M}_k^+}$ can produce a feasible solution in $O(kln)$. However, the quality of the solution degrades as we increase the number of sequences added to the original instance. A general method for reoptimization is applied to several self-reducible hard problems to further improve the approximability of the problem but with tradeoff on the running time of the algorithm. The PTAS for the CSP already provides the option of improving the approximation ratio by increasing the sampling size r in Algorithm 1. As r approaches t, we can have a solution with error that converges to 0 but with running time that is comparable to the naive exhaustive search that is exponential in

t. In the following section, we illustrate how reoptimization can help in improving the running time of the PTAS in [12].

4.3 Improving the PTAS for Reopt-CSP$_{\mathcal{M}_{k^+}}$

Let us consider an input instance $\mathcal{S}' = \{S_1, \ldots, S_t\}$ for CSP with a given optimal solution for a subset of its sequences. Without loss of generality, suppose we have an optimal solution OPT for the first r sequences in $\mathcal{S}'$, i.e., $\mathcal{S} = \{S_1, \ldots, S_r\}$. Let us also assume that $|\mathcal{S}' \setminus \mathcal{S}| = k$.

Algorithm 1 can provide a feasible solution for $\mathcal{S}'$ with an approximation ratio of $1 + \frac{4|\Sigma|-4}{\sqrt{e}(\sqrt{4r+1}-3)}$ in $O(l \cdot (tn)^{r+1})$. We argue in this section that we can achieve the same approximation ratio in $O(ltn((t-r)n)^r)$ time using the given OPT for $\mathcal{S}$, as presented in Theorem 3

In the following approximation algorithm, we implement the PTAS in Algorithm 1 by using the given optimal solution OPT.

Algorithm 4 Approximation algorithm for Reopt-$CSP_{\mathcal{M}_{k^+}}$ using the K-BEST-ALIGN procedure from Algorithm 3.

Input: Set of sequences $\mathcal{S}' = \{S_1, \ldots, S_t\}$, pattern length l, and optimal closest substring OPT of $\mathcal{S} = \{S_1, \ldots, S_r\}$
Output: Solution SOL' for $\mathcal{S}'$.

1: $SOL'_A =$ K-BEST-ALIGN($OPT, \{S_{r+1}, \ldots, S_t\}$)
2:
3: $SOL'_B = \emptyset$
4: $min = \infty$
5: **for each** r-sample $\in \{R(\mathcal{S}') \setminus R(\mathcal{S})\}$ **do**
6:    $SOL' =$ K-BEST-ALIGN($SOL, sample$)
7:    **if** $cost(SOL') < min$ **then**
8:        $min = cost(SOL')$
9:        $SOL'_B = SOL'$
10:    **end if**
11: **end for**
12:
13: **if** $cost(SOL'_A) < cost(SOL'_B)$ **then**
14:    **return** SOL'_A
15: **else**
16:    **return** SOL'_B
17: **end if**

THEOREM 3. *Algorithm 4 is a* $1 + \frac{4|\Sigma|-4}{\sqrt{e}(\sqrt{4r+1}-3)}$ *PTAS for* $Reopt\text{-}CSP_{\mathcal{M}_k^+}$ *which runs in* $O(ltn \cdot (t-r)n)^r)$.

PROOF. To prove that Algorithm 4 provides the same approximation ratio as Algorithm 1 from [12], we need to show that the set of all samples in $R(\mathcal{S}')$ is considered in Algorithm 4. SOL_A is obtained by aligning OPT to k other sequences in $\mathcal{S}'$ using Algorithm 2. Since we have the optimal solution for $\mathcal{S}$, $cost(SOL_A)$ is equal to the minimum over all feasible solutions sampled over $R(\mathcal{S})$. On the other hand, solution SOL_B is obtained by getting the minimum over all feasible solutions sampled from $R(\mathcal{S}') \setminus R(\mathcal{S})$. The output SOL' is the minimum between SOL_A and SOL_B. Thus, all samples from $R(\mathcal{S}')$ are considered in Algorithm 4.

For the running time of Algorithm 4, getting SOL_A will require $O(kn)$ steps. Line 5 will iterate in $O((tn)^r - (rn)^r)$. Getting the best alignment for each sample will require $O(tn)$ and getting the distance between two l-length substrings will take $O(l)$. Thus, Algorithm 4 runs in $O(ltn \cdot (t-r)n)^r)$. $\square$

5. CONCLUSION AND FUTURE WORK

In this paper, we presented reoptimization variant of the CSP under sequence addition. The additional information, i.e., optimal solution to a smaller instance, does not help to address the intractability of the problem, as shown in Theorem 2.

We considered the basic type of modification $\mathcal{M}_+$, where the new instance contains an additional sequence. The general approach in reoptimization is to transform the given optimal solution for it to become feasible to the new instance. For some cases, i.e., $v_{opt} = v'_{opt}$, our simple transformation can lead to OPT', otherwise we can have the worst case scenario where OPT cannot help to obtain OPT'. In fact, even for a small modification such as $\mathcal{M}_+$, v_{opt} can be totally different from v'_{opt}.

We generalized this approach by considering the modification $\mathcal{M}_{k+}$, characterized by adding k new sequences to the original instance. By using the same approach for Reopt-$\text{CSP}_{\mathcal{M}_+}$, we can obtain an error that grows as the number of additional sequences is increasing. However, we showed that by using the given optimal solution for reoptimization we can improve the running of the existing PTAS for CSP. With the same approximation ratio, we can improve the running time from $O((tn)^r)$ to $O(((t-r)n)^r)$.

It is natural to consider the opposite scenario, where an optimal solution to a larger instance is given and we are looking for the solution to a smaller subset of the instance. We argue that we are not also guaranteed to get the optimal solution by transforming the larger solution to provide a feasible solution to the smaller instance. Example 1 can also illustrate the idea. Exploring other techniques to make use of such kind of information is still part of our future work. Moreover, we would like to study other types of modification in the input instance such as changes in pattern length l.

6. ACKNOWLEDGEMENTS

7. REFERENCES

[1] Davide Bilò, Hans-Joachim Böckenhauer, Dennis Komm, Richard Královič, Tobias Mömke, Sebastian Seibert, and Anna Zych. Reoptimization of the shortest common superstring problem. *Algorithmica (New York)*, 61(2):227–251, 2011.

[2] Davide Bilo and Anna Zych. New Advances in Reoptimizing the Minimum Steiner Tree Problem. *In Proc. of the Mathematical Foundations of Computer Science, LNCS*, 7464:184–197, 2012.

[3] Hans-Joachim Böckenhauer, Juraj Hromkovič, Tobias Mömke, and Peter Widmayer. On the hardness of reoptimization. *In Proc. of the 34th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008), LNCS*, 4910, 2008.

[4] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Harold T. Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147(1-2):31–54, 1995.

[5] Nicolas Boria, Cécile Murat, and Vangelis Paschos. On the Probabilistic Min Spanning Tree Problem. *Journal of Mathematical Modelling and Algorithms*, 11(1):45–76, 2012.

[6] Brona Brejová, Daniel Brown, Ian M Harrower, and T Vinař. New Bounds for Motif Finding in Strong Instances. *In Proc. of the Combinatorial Pattern Matching*, pages 94–105, 2006.

[7] Riccardo Dondi, Guillaume Fertin, and Stéphane Vialette. Finding approximate and constrained motifs in graphs. *Theoretical Computer Science*, 483:10–21, April 2013.

[8] Patricia A. Evans, Andrew D. Smith, and H. Todd Wareham. On the complexity of finding common approximate substrings. *Theoretical Computer Science*, 306(1-3):407–430, September 2003.

[9] Michael R. Garey and David S. Johnson. A Guide to the Theory of NP-Completeness. *Computers and Intractibility*, January 1979.

[10] J. Kevin Lanctot, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Information and Computation*, 185(1):41–55, August 2003.

[11] Heng Li and Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11(5):473–483, September 2010.

[12] Ming Li, Bin Ma, and Lusheng Wang. Finding similar regions in many strings. *Proceedings of the thirty-first annual ACM symposium on Theory of computing - STOC '99*, 65(1):473–482, August 1999.

[13] Bin Ma. A Polynomial Time Approximation Scheme for the Closest Substring Problem. *Combinatorial Pattern Matching, Lecture Notes in Computer Science*, 1848:99–107, 2000.

[14] Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4955 LNBI(4):396–409, 2008.

[15] Pavel Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in DNA sequences. *Proceedings of the International Conference on Intelligent Systems for Molecular Biology ; ISMB. International Conference on Intelligent Systems for Molecular Biology*, 8:269–278, 2000.

[16] Markus W. Schäffter. Scheduling with forbidden sets.

Discrete Applied Mathematics, 72(1-2):155–166, 1997.

[17] Jianxin Wang, Jianer Chen, and Min Huang. An improved lower bound on approximation algorithms for the Closest Substring problem. *Information Processing Letters*, 107(1):24–28, June 2008.

[18] Elena Zaslavsky and Mona Singh. A combinatorial optimization approach for diverse motif finding applications. *Algorithms for molecular biology : AMB*, 1:13, January 2006.

[19] Anna Zych. *Reoptimization of NP-hard Problems.* PhD thesis, ETH Zurich, 2012.