Can You Solve Closest String Faster Than Exhaustive Search?

Amir Abboud ✉
Weizmann Institute of Science, Rehovot, Israel

Nick Fischer ✉
Weizmann Institute of Science, Rehovot, Israel

Elazar Goldenberg ✉
Academic College of Tel Aviv-Yaffo, Israel

Karthik C. S. ✉
Rutgers University, New Brunswick, NJ, USA

Ron Safier ✉
Weizmann Institute of Science, Rehovot, Israel

Abstract

We study the fundamental problem of finding the best string to represent a given set, in the form of the *Closest String* problem: Given a set $X \subseteq \Sigma^d$ of n strings, find the string x^* minimizing the radius of the smallest Hamming ball around x^* that encloses all the strings in X. In this paper, we investigate whether the Closest String problem admits algorithms that are faster than the trivial exhaustive search algorithm. We obtain the following results for the two natural versions of the problem:

- In the *continuous* Closest String problem, the goal is to find the solution string x^* anywhere in Σ^d. For binary strings, the exhaustive search algorithm runs in time $O(2^d \operatorname{poly}(nd))$ and we prove that it cannot be improved to time $O(2^{(1-\epsilon)d} \operatorname{poly}(nd))$, for any $\epsilon > 0$, unless the Strong Exponential Time Hypothesis fails.
- In the *discrete* Closest String problem, x^* is required to be in the input set X. While this problem is clearly in polynomial time, its fine-grained complexity has been pinpointed to be quadratic time $n^{2\pm o(1)}$ whenever the dimension is $\omega(\log n) < d < n^{o(1)}$. We complement this known hardness result with new algorithms, proving essentially that whenever d falls out of this hard range, the discrete Closest String problem can be solved faster than exhaustive search. In the small-d regime, our algorithm is based on a novel application of the inclusion-exclusion principle.

Interestingly, all of our results apply (and some are even stronger) to the natural dual of the Closest String problem, called the *Remotest String* problem, where the task is to find a string maximizing the Hamming distance to all the strings in X.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases Closest string, fine-grained complexity, SETH, inclusion-exclusion

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.3

Related Version *Full Version*: `https://arxiv.org/abs/2305.16878`

Funding *Amir Abboud*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme (grant agreement No 101078482). Additionally, Amir Abboud is supported by an Alon scholarship and a research grant from the Center for New Scientists at the Weizmann Institute of Science.
Karthik C. S.: This work was supported by a grant from the Simons Foundation, Grant Number 825876, Awardee Thu D. Nguyen and by the National Science Foundation under Grant CCF-2313372.

1 Introduction

The challenge of characterizing a set of strings by a single *representative* string is a fundamental problem all across computer science, arising in essentially all contexts where strings are involved. The basic task is to find a string x^* which minimizes the maximum number of mismatches to all strings in a given set X. Equivalently, the goal is to find the center x^* of a smallest ball enclosing all strings in X in the Hamming (or ℓ_0) metric. This problem has been studied under various names, including *Closest String*, *1-Center in the Hamming metric* and *Chebyshev Radius*, and constitutes the perhaps most elementary clustering task for strings.

In the literature, the Closest String problem has received a lot of attention [11, 12, 23, 14, 22, 26, 9, 25, 20, 27, 1], and it is not surprising that besides the strong theoretical interest, it finds wide-reaching applications in various domains including machine learning, bioinformatics, coding theory and cryptography. One such application in machine learning is for clustering *categorical* data. Typical clustering objectives involve finding good center points to characterize a set of feature vectors. For numerical data (such as a number of publications) this task translates to a center (or median) problem over, say, the ℓ_1 metric which can be solved using geometry tools. For categorical data, on the other hand, the points have non-numerical features (such as blood type or nationality) and the task becomes finding a good center string over the Hamming metric.

Another important application, in the context of computational biology, is the computer-aided design of PCR primers [25, 24, 10, 29, 13, 32]. On a high level, in the PCR method the goal is to find and amplify (i.e., copy millions of times) a certain fragment of some sample DNA. To this end, short DNA fragments (typically 18 to 25 nucleotides) called *primers* are used to identify the start and end of the region to be copied. These fragments should match as closely as possible the target regions in the sample DNA. Designing such primers is a computational task that reduces exactly to finding a closest string in a given set of genomes.

The Closest String problem comes in two different flavors: In the *continuous* Closest String problem the goal is to select an *arbitrary* center string $x^* \in \Sigma^d$ (here, Σ is the underlying alphabet) that minimizes the maximum Hamming distance to the n strings in X. This leads to a baseline algorithm running in exponential time $O(|\Sigma|^d \operatorname{poly}(nd))$. In the *discrete* Closest String problem, in contrast, the task is to select the best center x^* in the given set of strings X; this problem therefore admits a baseline algorithm in time $O(n^2 d)$. Despite the remarkable attention that both variants have received so far, the most basic questions about the continuous and discrete Closest String problems have not been fully resolved yet:

Can the $O(|\Sigma|^d \operatorname{poly}(nd))$-time algorithm for continuous Closest String be improved?
Can the $O(n^2 d)$-time algorithm for discrete Closest String be improved?

In this paper, we make considerable progress towards resolving both driving questions, by respectively providing tight conditional lower bounds and new algorithms. In the upcoming Sections 1.1 and 1.2 we will address these questions in depth and state our results.

Interestingly, in both cases our results also extend, at times even in a stronger sense, to a natural dual of the Closest String problem called the *Remotest String* problem. Here, the task is to find a string x^* that maximizes the minimum Hamming distance from x^* to a given set of strings X. This problem has also been studied in computational biology [22, 21] and more prominently in the context of coding theory: The remotest string distance is a fundamental parameter of any code which is also called the *covering radius* [8], and under this name the Remotest String problem has been studied in previous works [28, 15, 6, 17] mostly for specific sets X such as linear codes or lattices. See Alon, Panigrahy and Yekhanin [6] for further connections to matrix rigidity.

1.1 Continuous Closest/Remotest String

Let us start with the more classical *continuous* Closest String problem. It is well-known that the problem is NP-complete [11, 22], and up to date the best algorithm remains the naive one: Exhaustively search through all possible strings in time $O(|\Sigma|^d \operatorname{poly}(nd))$. This has motivated the study of approximation algorithms leading to various approximation schemes [12, 23, 25, 27], and also the study through the lens of parameterized algorithms [14]. In this work, we insist on exact algorithms and raise again the question: *Can you solve the continuous Closest String problem faster than exhaustive search?*

For starters, focus on the Closest String problem for *binary* alphabets (i.e., for $|\Sigma| = 2$) which is of particular importance in the context of coding theory [20]. From the known NP-hardness reduction which is based on the 3-SAT problem [11], it is not hard to derive a $2^{d/2}$ lower bound under the Strong Exponential Time Hypothesis (SETH) [18, 19]. This bound clearly does not match the upper bound and possibly leaves hope for a meet-in-the-middle-type algorithm. In our first contribution we shatter all such hopes by strengthening the lower bound, with considerably more effort, to match the time complexity of exhaustive search:

▶ **Theorem 1** (Continuous Closest String is SETH-Hard). *The continuous Closest String problem cannot be solved in time $O(2^{(1-\epsilon)d} \operatorname{poly}(n))$, for any $\epsilon > 0$, unless SETH fails.*

Interestingly, we obtain this lower bound as a corollary of the analogous lower bound for the continuous Remotest String problem (see the following Theorem 2). This is because both problems are equivalent over the binary alphabet. However, even for larger sized alphabet sets Σ, we obtain a matching lower bound against the Remotest String problem:

▶ **Theorem 2** (Continuous Remotest String is SETH-Hard). *The continuous Remotest String problem cannot be solved in time $O(|\Sigma|^{(1-\epsilon)d} \operatorname{poly}(n))$, for any $\epsilon > 0$ and $|\Sigma| = o(d)$, unless SETH fails.*

Theorem 2 gives a *tight* lower bound for the continuous Remotest String problem in all regimes where we can expect lower bounds, and we therefore close the exact study of the continuous Remotest String problem. Indeed, in the regime where the alphabet size $|\Sigma|$ exceeds the dimension d, the Closest and Remotest String problems *can* be solved faster in time $O(d^d \operatorname{poly}(n, d))$ (and even faster parameterized in terms of the target distance [14]).

The intuition behind Theorem 2 is simple: We encode a k-SAT instance as a Remotest String problem by viewing strings as *assignments* and by searching for a string which is remote from all *falsifying* assignments. The previously known encoding [11] was inefficient (encoding a single variable X_i accounted for *two* letters in the Remotest String instance: one for encoding the truth value and another one as a “don’t care” value for clauses not containing X_i), and our contribution is that we make the encoding lossless. While superficially simple, this baseline idea requires a lot of technical effort.

1.2 Discrete Closest/Remotest String

Recall that in the *discrete* Closest String problem (in contrast to the continuous one) the solution string x^* must be part of the input set X. For applications in the context of data compression and summarization, the discrete problem is often the better choice: Selecting the representative string from a set of, say, grammatically or semantically meaningful strings is typically more informative than selecting an arbitrary representative string.

The problem can be naively solved in time $O(n^2 d)$ by exhaustive search: Compute the Hamming distance between all $\binom{n}{2}$ pairs of strings in X in time $O(d)$ each. In terms of exact algorithms, this running time is the fastest known. Toward our second driving question, we investigate whether this algorithm can be improved, at least for some settings of n and d. In previous work, Abboud, Bateni, Cohen-Addad, Karthik, and Seddighin [1] have established a conditional lower bound under the Hitting Set Conjecture [3], stating that the problem requires quadratic time in n whenever $d = \omega(\log n)$:

► **Theorem 3** (Discrete Closest String for Super-Logarithmic Dimensions [1]). *The discrete Closest String problem in dimension $d = \omega(\log n)$ cannot be solved in time $O(n^{2-\epsilon})$, for any $\epsilon > 0$, unless the Hitting Set Conjecture fails.*

This hardness result implies that there is likely no polynomially faster algorithm for Closest String whenever the dimension d falls in the range $\omega(\log n) < d < n^{o(1)}$. But this leaves open the important question of whether the exhaustive search algorithm can be improved outside this region, if d is very small (say, $o(\log n)$) or very large (i.e., polynomial in n). In this paper, we provide answers for both regimes.

Small Dimension. Let us start with the small-dimension regime, $d = o(\log n)$. The outcome of the question whether better algorithms are possible is a priori not clear. Many related center problems (for which the goal is to select a center point x^* that is closest not necessarily in the Hamming metric but in some other metric space) differ substantially in this regard: On the one hand, in the Euclidian metric, even for $d = 2^{O(\log^* n)}$, the center problem requires quadratic time under the Hitting Set Conjecture [1].[1] On the other hand, in stark contrast, the center problem for the ℓ_1 and ℓ_∞ metrics can be solved in almost-linear time $n^{1+o(1)}$ whenever the dimension is $d = o(\log n)$. This dichotomy phenomenon extends to even more general problems including nearest and furthest neighbor questions for various metrics and the maximum inner product problem [33, 7].

In view of this, we obtain the perhaps surprising result that whenever $d = o(\log n)$ the discrete Closest String problem can indeed be solved in subquadratic – even almost-linear – time. More generally, we obtain the following algorithm:

► **Theorem 4** (Discrete Closest String for Small Dimensions). *The discrete Closest String problem can be solved in time $O(n \cdot 2^d)$.*

Note that this result is trivial for binary alphabets, and our contribution lies in finding an algorithm in time $O(n \cdot 2^d)$ for alphabets of *arbitrary* size.

We believe that this result is interesting also from a technical perspective, as it crucially relies on the *inclusion-exclusion* principle. While this technique is part of the everyday tool-set for exponential-time and parameterized algorithms, it is uncommon to find applications for polynomial-time problems and our algorithm yields the first such application to a center-type

[1] Technically, the problem is only known to be hard in the *listing* version where we require to list *all* feasible centers [1].

problem, to the best of our knowledge. We believe that our characterization of the Hamming distance in terms of an inclusion-exclusion-type formula (see Lemma 23) is very natural and likely to find applications in different contexts.

Large Dimension. In the large-dimension regime, where d is polynomial in n, it is folklore that fast matrix multiplication should be of use. Specifically, over a binary alphabet we can solve the Closest String problem in time $O(\mathrm{MM}(n, d, n))$ (where $\mathrm{MM}(n, d, n)$ is the time to multiply an $n \times d$ by a $d \times n$ matrix) by using fast matrix multiplication to compute the Hamming distances between all pairs of vectors, rather than by brute-force. For arbitrary alphabet sizes this idea leads to a running time of $O(\mathrm{MM}(n, d|\Sigma|, n))$ which is of little use as $|\Sigma|$ can be as large as n and in this case the running time becomes $\Omega(n^2 d)$.

We prove that nevertheless, the $O(n^2 d)$-time baseline algorithm can be improved using fast matrix multiplication – in fact, using ideas from *sparse* matrix multiplication such as Yuster and Zwick's heavy-light idea [34].

▶ **Theorem 5** (Discrete Closest String for Large Dimensions). *For all $\delta > 0$, there is some $\epsilon > 0$ such that the discrete Closest String problem with dimension $d = n^\delta$ can be solved in time $O(n^{2+\delta-\epsilon})$.*

Remotest String. Finally, we turn our attention to the discrete Remotest String problem. In light of the previously outlined equivalence in the continuous setting, we would expect that also in the discrete setting, the Closest and Remotest String problem are tightly connected. We confirm this suspicion and establish a strong equivalence for binary alphabets:

▶ **Theorem 6** (Equivalence of Discrete Closest and Remotest String). *If the discrete Closest String over a binary alphabet is in time $T(n, d)$, then the discrete Remotest String over a binary alphabet is in time $T(O(n), O(d + \log n)) + \widetilde{O}(nd)$. Conversely, if the discrete Remotest String over a binary alphabet is in time $T'(n, d)$, then the discrete Closest String over a binary alphabet is in time $T'(O(n), O(d + \log n)) + \widetilde{O}(nd)$.*

In combination with Theorem 3, this equivalence entails that also Remotest String requires quadratic time in the regime $\omega(\log n) < d < n^{o(1)}$. Let us remark that, while the analogous equivalence is trivial in the continuous regime, proving Theorem 6 is not trivial and involves the construction of a suitable gadget that capitalizes on explicit constant-weight codes.

The similarity between discrete Closest and Remotest String continues also on the positive side: All of our algorithms extend naturally to Remotest String, not only for binary alphabets (see the full version for more details).

1.3 Open Problems

Our work inspires some interesting open problems. The most pressing question from our perspective is whether there also is a $|\Sigma|^{(1-o(1))d}$ lower-bound for continuous Closest String (for alphabets of size bigger than 2).

▶ **Open Question 7** (Continuous Closest String for Large Alphabets). *For $|\Sigma| > 2$, can the continuous Closest String problem be solved in time $O(|\Sigma|^{(1-\epsilon)d} \operatorname{poly}(n))$, for some $\epsilon > 0$?*

We believe that our approach (proving hardness under SETH) hits a natural barrier for the Closest String problem. In some sense, the k-SAT problem behaves very similarly to Remotest String (with the goal to be remote from all falsifying assignments), and over binary alphabets remoteness and closeness can be exchanged. For larger alphabets this trivial equivalence simply does not hold. It would be exciting if this insight could fuel a faster *algorithm* for Closest String, and we leave this question for future work.

On the other hand, consider again the discrete Closest and Remotest String problems. While we close *almost all* regimes of parameters, there is one regime which we did not address in this paper:

▶ **Open Question 8** (Discrete Closest/Remotest String for Logarithmic Dimension). *Let c be a constant. Can the discrete Closest and Remotest String problems with dimension $d = c \log n$ be solved in time $O(n^{2-\epsilon})$, for some $\epsilon = \epsilon(c) > 0$?*

In the regime $d = \Theta(\log n)$, we typically expect only very sophisticated algorithms, say using the polynomial method in algorithm design [2], to beat exhaustive search. And indeed, using the polynomial method it is possible to solve also discrete Closest and Remotest String in subquadratic time for binary (or more generally, constant-size) alphabets [5, 4, Theorem 1.4]. The question remains whether subquadratic time complexity is also possible for unrestricted alphabet sizes.

1.4 Outline of the Paper

We organize this paper as follows. In Section 2 we give some preliminaries and state the formal definitions of the continuous/discrete Closest/Remotest String problems. In Section 3 we prove our conditional hardness results for the continuous problems. In Section 4 we treat in detail the discrete problems. Throughout, due to space constraints, we defer several proofs to the full version of this paper.

2 Preliminaries

We set $[n] = \{1, \dots, n\}$ and write $\widetilde{O}(T) = T(\log T)^{O(1)}$ and $\mathrm{poly}(n) = n^{o(1)}$. We occasionally write $\mathbf{1}(P) \in \{0, 1\}$ to express the truth value of the proposition P.

Strings. Let Σ be a finite alphabet of size at least 2. For a string $x \in \Sigma^d$ of length (or dimension) d, we write $x[i]$ for the i-th character in x. For a subset $I \subseteq [d]$, we write $x[I] \in \Sigma^I$ for the subsequence obtained from x by restricting to the characters in I. The *Hamming distance* between two equal-length strings $x, y \in \Sigma^d$ is defined as $\mathrm{HD}(x, y) = |\{i \in [d] : x[i] \neq y[i]\}|$. Let X be a set of length-d strings and let x^* be a length-d string. Then we set

$$r(x^*, X) = \max_{y \in X} \mathrm{HD}(x^*, y) \quad \text{(the } \textit{radius} \text{ of } X \text{ around } x^*\text{)},$$

$$d(x^*, X) = \min_{y \in X} \mathrm{HD}(x^*, y) \quad \text{(the } \textit{distance} \text{ from } x^* \text{ to } X\text{)}.$$

Let us formally repeat the definitions of the four problems studied in this paper:

▶ **Definition 9** (Continuous Closest String). *Given a set of n strings $X \subseteq \Sigma^d$, find a string $x^* \in \Sigma^d$ which minimizes the radius $r(x^*, X)$.*

▶ **Definition 10** (Continuous Remotest String). *Given a set of n strings $X \subseteq \Sigma^d$, find a string $x^* \in \Sigma^d$ which maximizes the distance $d(x^*, X)$.*

▶ **Definition 11** (Discrete Closest String). *Given a set of n strings $X \subseteq \Sigma^d$, find a string $x^* \in X$ which minimizes the radius $r(x^*, X)$.*

▶ **Definition 12** (Discrete Remotest String). *Given a set of n strings $X \subseteq \Sigma^d$, find a string $x^* \in X$ which maximizes the distance $d(x^*, X \setminus \{x^*\})$.*

Hardness Assumptions. In this paper, our lower bounds are conditioned on the following two plausible hypotheses from fine-grained complexity.

▶ **Definition 13** (Strong Exponential Time Hypothesis, SETH [18, 19])**.** *For all $\epsilon > 0$, there is some $k \geq 1$ such that k-CNF SAT cannot be solved in time $O(2^{(1-\epsilon)n})$.*

▶ **Definition 14** (Hitting Set Conjecture [3])**.** *For all $\epsilon > 0$, there is some $c \geq 1$ such that no algorithm can decide in $O(n^{2-\epsilon})$ time, whether in two given lists A, B of n subsets of a universe of size $c \log n$, there is a set in the first list that intersects every set in the second list (i.e. a "hitting set").*

3 Continuous Closest String is SETH-Hard

In this section we present our fine-grained lower bounds for the continuous Closest and Remotest String problems. We start with a high-level overview of our proof, and then provide the technical details in Sections 3.1–3.4.

Let us first recall that over *binary* alphabets, the continuous Closest and Remotest String problems are trivially equivalent. The insight is that for any two strings $x, y \in \{0,1\}^d$ we have that $\mathrm{HD}(x, y) = d - \mathrm{HD}(\overline{x}, y)$ where $\overline{x}$ is the *complement* of x obtained by flipping each bit. From this it easily follows that

$$\min_{x^* \in \{0,1\}^d} \max_{y \in X} \mathrm{HD}(x^*, y) = d - \max_{x^* \in \{0,1\}^d} \min_{y \in X} \mathrm{HD}(x^*, y).$$

Note that finding a string x^* optimizing the left-hand side is exactly the Closest String problem, whereas finding a string x^* optimizing the right-hand side is exactly the Remotest String problem, and thus both problems are one and the same. For this reason, let us focus our attention for the rest of this section only on the Remotest String problem.

Tight Lower Bound for Remotest String. Our goal is to establish a lower bound under the Strong Exponential Time Hypothesis. To this end, we reduce a k-SAT instance with N variables to an instance of the Remotest String problem with dimension $d = (1 + o(1))N$. In Sections 3.1–3.4 we will actually reduce from a q-ary analogue of the k-SAT in order to get a tight lower bound for all alphabet sizes $|\Sigma|$. However, for the sake of simplicity we stick to binary strings and the usual k-SAT problem in this overview. Our reduction runs in two steps.

Step 1: Massaging the SAT Formula. In the first step, we bring the given SAT formula into a suitable shape for the reduction to the Remotest String problem. Throughout, we partition the variables $[N]$ into *groups* $P_1, \dots, P_{\frac{N}{s}}$ of size exactly s (where s is a parameter to be determined later). We assert the following properties:

- *Regularity:* All clauses contain exactly k literals, and all clauses contain literals from the same number of groups (say r). This property can be easily be guaranteed by adding a few fresh variables to the formula, all of which must be set to 0 in a satisfying assignment, and by adding these variables to all clauses which do not satisfy the regularity constraint yet.
- *Balancedness:* Let us call an assignment $\alpha \in \{0,1\}^N$ *balanced* if in every group it assigns exactly half the variables to 0 and half the variables to 1. We say that a formula is *balanced* if it is either unsatisfiable or if it is satisfiable by a balanced assignment. To make sure that a given formula is balanced, we can for instance flip each variable in the

formula with probability $\frac{1}{2}$. In this way we balance each group with probability $\approx \frac{1}{\sqrt{s}}$, and so all $\frac{N}{s}$ groups are balanced with probability at least $s^{-\frac{N}{2s}}$. By choosing $s = \omega(1)$, this random experiment yields a balanced formula after a negligible number of repetitions. In Lemma 19 we present a deterministic implementation of this idea.

Step 2: Reduction to Remotest String. The next step is to reduce a regular and balanced k-CNF formula to an instance of the Remotest String problem. The idea is to encode all *falsifying* assignments of the formula as strings – a sufficiently remote point should in spirit be remote from falsifying and thus satisfying. To implement this idea, take any clause C from the instance. Exploiting the natural correspondence between strings and assignments, we add all strings $\alpha \in \{0,1\}^n$ that satisfy the following two constraints to the Remotest String instance:

1. The assignment α falsifies the clause C.
2. For any group P_i that does *not* contain a variable from C, we have that $\alpha[P_i] = 0^s$ or $\alpha[P_i] = 1^s$.

We start with the intuition behind the second constraint: For any *balanced* assignment α and any group P_i that does not contain a variable from C, we have that $\mathrm{HD}(\alpha^*[P_i], \alpha[P_i]) = \frac{s}{2}$ (the string $\alpha^*[P_i]$ contains half zeros and half ones, whereas $\alpha[P_i]$ is either all-zeros or all-ones). There are exactly $\frac{N}{s} - r$ such groups (by the regularity), leading to Hamming distance $\frac{s}{2}(\frac{N}{s} - r)$.

It follows that the only groups that actually matter for the distance between α^* and α are the groups which do contain a variable from C. Here comes the first constraint into play: If α^* is a satisfying assignment, then α^* and α must differ in at least one of these groups and therefore have total distance at least $\frac{s}{2}(\frac{N}{s} - r) + 1$. Conversely, for any falsifying assignment α^* there is some string α in the instance with distance at most $\frac{s}{2}(\frac{N}{s} - r)$. Therefore, to decide whether the SAT formula is satisfiable it suffices to compute whether there is a Remotest String with distance at least $\frac{s}{2}(\frac{N}{s} - r) + 1$. Finally, it can be checked that the number of strings α added to the instance is manageable.

This completes the outline of our hardness proof, and we continue with the details. In Section 3.1 we introduce the (q,k)-SAT problem which we will use to give a clean reduction also for alphabet larger than size 2. In Section 3.2 we formally prove how to guarantee that a given (q,k)-SAT formula is regular and balanced, and in Section 3.3 we give the details about the reduction to the Remotest String problem. We put these pieces together in Section 3.4 and formally prove Theorem 2.

3.1 q-ary SAT

To obtain our full hardness result, we base our reduction on the hardness of *q-ary* analogue of the classical k-SAT problem. We start with an elaborate definition of this problem. Let $X_1, \dots, X_N$ denote some q-ary variables (i.e., variables taking values in the domain $[q]$). A *literal* is a Boolean predicate of the form $X_i \neq a$, where x_i is one of the variables and $a \in [q]$. A *clause* is a disjunction of several literals; we say the clause has *width* k if it contains exactly k literals. A *(q,k)-CNF formula* is a disjunction of clauses of width at most k. Finally, in the (q,k)-SAT problem, we are given a (q,k)-CNF formula over M clauses and N q-ary variables, and the task is to check whether there exists an assignment $\alpha \in [q]^N$ which satisfies all clauses. This problem has already been addressed in previous works [31, 30], and it is known that q-ary SAT cannot be solved faster than exhaustive search unless SETH fails:

▶ **Lemma 15** (q-ary SAT is SETH-Hard [30, Theorem 3.3]). *For any $\epsilon > 0$, there is some $k \geq 3$ such that for all $q = q(N) \geq 2$, (q,k)-SAT cannot be solved in time $O(q^{(1-\epsilon)N} \operatorname{poly}(M))$, unless SETH fails.*

While k is always constant, note that this hardness result applies even when q grows with N. We will later exploit this by proving hardness for Remotest String even for alphabets of super-constant size.

3.2 Regularizing and Balancing

Before we get to the core of our hardness result, we need some preliminary lemmas on the structure of (q,k)-CNF formulas. Throughout, let N be the number of variables and let $\mathcal{P}$ be a partition of N into *groups* of size exactly s. (Note that the existence of P implies that N is divisible by s.) In two steps we will now formally introduce the definitions of regular and balanced formulas and show how to convert unconstrained formulas into regular and balanced ones. We defer the proofs of the upcoming lemmas to the full version of this paper.

▶ **Definition 16** (Regular Formulas). *Let ϕ be a (q,k)-CNF formula over N variables, and let $\mathcal{P}$ be a partition of $[N]$. We say that ϕ is* r-regular *(with respect to $\mathcal{P}$) if every clause contains exactly k literals from exactly r distinct groups in $\mathcal{P}$.*

▶ **Lemma 17** (Regularizing). *Let ϕ be a (q,k)-CNF formula, and let $2k \leq s \leq N$. In time $\operatorname{poly}(NM)$ we can construct a $(q,2k)$-CNF formula ϕ' satisfying the following properties:*

- *ϕ' is satisfiable if and only if ϕ is satisfiable.*
- *ϕ' has at most $N + O(s)$ variables and at most $M + O(s \operatorname{poly}(q))$ clauses.*
- *ϕ' is $(k+1)$-regular with respect to some partition $\mathcal{P}$ into groups of size exactly s.*

▶ **Definition 18** (Balanced Formulas). *Let $\mathcal{P}$ be a partition of $[N]$ into groups of size s. We say that an assignment $\alpha \in [q]^N$ is* balanced *(with respect to $\mathcal{P}$) if in every group of $\mathcal{P}$, α assigns each symbol in $[q]$ exactly $\frac{s}{q}$ times. We say that a (q,k)-CNF formula ϕ is* balanced *(with respect to $\mathcal{P}$) if either ϕ is unsatisfiable, or ϕ is satisfiable by a balanced assignment α.*

▶ **Lemma 19** (Balancing). *Let ϕ be a (q,k)-CNF formula over N variables, let $\mathcal{P}$ be a partition of $[N]$ into groups of size s, and assume that q divides s. We can construct (q,k)-CNF formulas $\phi_1, \dots, \phi_t$ over the same number of variables and clauses as ϕ such that:*

- *For all $i \in [t]$, ϕ_i is satisfiable if and only if ϕ is satisfiable.*
- *There is some $i \in [t]$ such that ϕ_i is balanced (with respect to $\mathcal{P}$).*
- *$t = ((s+1)(q-1))^{(q-1)\frac{N}{s}}$, and we can construct each formula in time $\operatorname{poly}(NMt)$.*

3.3 Reduction to Remotest String

Having in mind that for our reduction we can assume the SAT formula to be regular and balanced, the following lemma constitutes the core of our reduction:

▶ **Lemma 20** (Reduction from Regular Balanced SAT to Remotest String). *Suppose there is an algorithm for the continuous Remotest String problem, running in time $O(|\Sigma|^{(1-\epsilon)d} \operatorname{poly}(n))$, for some $\epsilon > 0$. Then there is an algorithm that decides whether a given s-partitioned r-regular (q,k)-SAT formula is satisfiable, and runs in time $O(q^{(1-\epsilon)N + O(s + \frac{N}{s})} \operatorname{poly}(M))$.*

Proof. We start with some notation: For a clause C, we write $\mathcal{P}(C) \subseteq \mathcal{P}$ to address all groups containing a literal from C. We start with the construction of the Remotest String instance with alphabet $\Sigma = [q]$ and dimension $d = N$. Here, we make use of the natural correspondence between strings $\alpha \in \Sigma^d$ and assignments $\alpha \in [q]^N$. In the instance, we add the following strings: For each clause C, add all assignments $\alpha \in [q]^N$ to the instance which satisfy the following two constraints:

1. The assignment α falsifies the clause C.
2. For each group $P \in \mathcal{P} \setminus \mathcal{P}(C)$, the subsequence $\alpha[P]$ contains only one symbol. (That is, $\alpha[P] = a^s$ for some $a \in [q]$.)

We prove that this instance is complete and sound.

▷ Claim 21 (Completeness). If ϕ is satisfiable, then there is some $\alpha^* \in [q]^d$ with $d(\alpha^*, X) > \frac{(q-1)(N-rs)}{q}$.

Proof. Since we assume that the formula ϕ is satisfiable and balanced, there is a satisfying and balanced assignment α^*. To prove that $d(\alpha^*, X) > \frac{(q-1)(N-rs)}{q}$, we prove that for each string α added to the Remotest String instance, we have $\mathrm{HD}(\alpha^*, \alpha) > \frac{(q-1)(N-rs)}{q}$. Let C be the clause associated to α. From the two conditions on α, we get the following two bounds.

By the first condition, α is a falsifying assignment of C. In particular, the subsequence $\alpha[\bigcup_{P\in\mathcal{P}(C)} P]$ falsifies C (which is guaranteed to contain all variables visible to C) falsifies C. Since α^* is a satisfying assignment to the whole formula, and in particular to C, we must have that $\alpha^*[\bigcup_{P\in\mathcal{P}(C)} P] \neq \alpha[\bigcup_{P\in\mathcal{P}(C)} P]$, and thus $\sum_{P\in\mathcal{P}(C)} \mathrm{HD}(\alpha^*[P], \alpha[P]) \geq 1$.

By the second condition, for any group $P \in \mathcal{P} \setminus \mathcal{P}(C)$, the subsequence $\alpha[P]$ contains only one symbol. Since α^* is balanced, $\alpha^*[P]$ contains that symbol exactly in a $1/q$-fraction of the positions and differs in the remaining ones from $\alpha[P]$. It follows that $\mathrm{HD}(\alpha^*[P], \alpha[P]) = s - \frac{s}{q} = \frac{(q-1)s}{q}$.

Combining both bounds, we have that

$$\begin{aligned}\mathrm{HD}(\alpha^*, \alpha) &= \sum_{P\in\mathcal{P}(C)} \mathrm{HD}(\alpha^*, \alpha[P]) + \sum_{P\in\mathcal{P}\setminus\mathcal{P}(C)} \mathrm{HD}(\alpha^*, \alpha[P]) \\ &\geq 1 + \left(\frac{N}{s} - r\right) \cdot \frac{(q-1)s}{q} = \frac{(q-1)(N-rs)}{q} + 1,\end{aligned}$$

and the claim follows. ◁

▷ Claim 22 (Soundness). If ϕ is not satisfiable, then for all $\alpha^* \in [q]^d$ we have $d(\alpha^*, X) \leq \frac{(q-1)(N-rs)}{q}$.

Proof. Take any $\alpha^* \in [q]^d$. Since ϕ is not satisfiable, α^* is a falsifying assignment of ϕ and thus there is some clause C that is falsified by α^*. Our strategy is to find some string $\alpha \in [q]^d$ in the constructed instance with $\mathrm{HD}(\alpha^*, \alpha) \leq \frac{(q-1)(N-rs)}{q}$.

We define that string α group-wise: In the groups $\mathcal{P}(C)$ touching C, we define α to be exactly as α^*, that is, $\alpha[\bigcup_{P\in\mathcal{P}(C)}] := \alpha^*[\bigcup_{P\in\mathcal{P}(C)}]$. For each group $P \in \mathcal{P} \setminus \mathcal{P}(C)$ not touching C, let $a \in [q]$ be an arbitrary symbol occurring at least $\frac{s}{q}$ times in $\alpha^*[P]$ and assign $\alpha[P] := a^s$. By this construction we immediately have $\mathrm{HD}(\alpha[P], \alpha^*[P]) \leq s - \frac{s}{q} = \frac{(q-1)s}{q}$, and in total

$$\begin{aligned}\mathrm{HD}(\alpha^*, \alpha) &= \sum_{P\in\mathcal{P}(C)} \mathrm{HD}(\alpha^*, \alpha[P]) + \sum_{P\in\mathcal{P}\setminus\mathcal{P}(C)} \mathrm{HD}(\alpha^*, \alpha[P]) \\ &\leq 0 + \left(\frac{N}{s} - r\right) \cdot \frac{(q-1)s}{q} = \frac{(q-1)(N-rs)}{q},\end{aligned}$$

as claimed. ◁

In combination, Claims 21 and 22 show that the constructed instance of the Remotest String problem is indeed equivalent to the given (q, k)-SAT instance ϕ in the sense that ϕ is satisfiable if and only if there is a remote string with distance more than $\frac{(q-1)(N-rs)}{q}$.

It remains to analyze the running time. Let n denote the number of strings in the constructed instance. As a first step, we prove that $n \le q^{O(s)+\frac{N}{s}} \cdot M$ and that we can construct the instance in time $\mathrm{poly}(n)$. Indeed, focus on any clause C. The strings α in the instance are unconstrained in all groups touching C (up to the condition that α must falsify C) which accounts for $r \cdot s$ positions and thus $q^{rs} = q^{O(s)}$ options. For each group not touching C we can choose between q possible values, and therefore the total number of options is $q^{\frac{N}{s}-r} \le q^{\frac{N}{s}}$. Therefore, the total number of strings is indeed $n \le M \cdot q^{O(s)} \cdot q^{\frac{N}{s}}$. Moreover, it is easy to see that the instance can be constructed in time $\mathrm{poly}(n)$.

As the time to construct the instance is negligible, the total running time is dominated by solving the Remotest String instance. Assuming an algorithm in time $O(|\Sigma|^{(1-\epsilon)d} \mathrm{poly}(n))$, this takes time $O(q^{(1-\epsilon)N+O(s+\frac{N}{s})} \mathrm{poly}(M))$ as claimed. ◀

3.4 Putting the Pieces Together

We are finally ready to prove Theorems 1 and 2.

▶ **Theorem 2** (Continuous Remotest String is SETH-Hard). *The continuous Remotest String problem cannot be solved in time $O(|\Sigma|^{(1-\epsilon)d} \mathrm{poly}(n))$, for any $\epsilon > 0$ and $|\Sigma| = o(d)$, unless SETH fails.*

Proof. Suppose that the continuous Remotest String problem is in time $O(|\Sigma|^{(1-\epsilon)d} \mathrm{poly}(n))$ for some $\epsilon > 0$ and for $|\Sigma| = o(d)$. With this in mind, we design a better-than-brute-force (q, k)-SAT algorithm for $q = |\Sigma|$ by combining the previous three Lemmas 17, 19, and 20. Let ϕ be the input formula, and let $\mathcal{P}$ denote a partition of the variables into groups of size s (which is yet to be determined) as before.

1. Using Lemma 17, construct a *regular* $(q, 2k)$-formula ϕ' which is equivalent to ϕ.
2. Using Lemma 19, construct regular $(q, 2k)$-formulas $\phi'_1, \dots, \phi'_t$ all of which are equivalent to ϕ. At least one of these formulas is *balanced*.
3. By means of the reduction in Lemma 20, solve all t formulas $\phi'_1, \dots, \phi'_t$. If a formula is reported to be satisfiable, check whether the answer is truthful (e.g., using the standard decision-to-reporting reduction) and if so report that the formula is satisfiable. We need the additional test since, strictly speaking, we have not verified in Lemma 20 that the algorithm is correct for non-balanced inputs.

The correctness is obvious. Let us analyze the running time. Constructing the formula ϕ' takes polynomial time and can be neglected. By Lemma 17, ϕ' has $N' = N + O(s)$ variables and $M' = M + O(s \,\mathrm{poly}(q))$ clauses. The construction of the formulas $\phi'_1, \dots, \phi'_t$ also runs in polynomial time $\mathrm{poly}(N'M't)$ and can be neglected; this time we do not increase the number of variables and clauses. Moreover, Lemma 19 guarantees that

$$t = ((s+1)(q-1))^{(q-1)\frac{N'}{s}} \le (sq)^{O(\frac{qN}{s})},$$

By picking $s = cq$ (for some parameter c to be determined), this becomes

$$t \le (cq^2)^{O(\frac{N}{c})} = q^{O(\frac{N}{c}\log_q(cq^2))} = q^{N \cdot O(\frac{\log c}{c})}.$$

Finally, by Lemma 20 solving each formula ϕ'_i takes time

$$q^{(1-\epsilon)N'+O(s+\frac{N'}{s})} \mathrm{poly}(M') = q^{(1-\epsilon)N+O(s+\frac{N}{s})} \mathrm{poly}(M) = q^{(1-\epsilon)N+o(cN)+O(\frac{N}{c})} \mathrm{poly}(M),$$

(using that $s = cq = o(cN)$), and thus the total running time is bounded by

$$q^{N \cdot O(\frac{\log c}{c})} \cdot q^{(1-\epsilon)N + o(cN) + O(\frac{N}{c})} \operatorname{poly}(M) = q^{(1-\epsilon+o(c)+O(\frac{\log c}{c}))N} \operatorname{poly}(M).$$

Note that by picking c to be a sufficiently large constant (depending on ϵ), the exponent becomes $(1 - \frac{\epsilon}{2})N$, say. We have therefore obtained an algorithm for the (q, k)-SAT problem in time $O(q^{(1-\epsilon/2)N} \operatorname{poly}(M))$, which contradicts SETH by Lemma 15. ◀

4 Discrete Closest String via Inclusion-Exclusion

In this section, we present an algorithm for the discrete Closest String problem with *subquadratic* running time whenever the dimension is small, i.e. $d = o(\log n)$. Our algorithm relies on the inclusion-exclusion principle, and is, to the best of our knowledge, the first application of this technique to the Closest and Remotest String problems. Specifically, we obtain the following result:

▶ **Theorem 4** (Discrete Closest String for Small Dimensions). *The discrete Closest String problem can be solved in time $O(n \cdot 2^d)$.*

We structure this section as follows: First, we present a high-level overview of the main ideas behind the algorithm; for the sake of presentation, we focus only on the Closest String problem. We start developing a combinatorial toolkit to tackle the Closest String problem (with all proofs deferred to the full version of this paper). Then, in Section 4.1 we provide the actual algorithm and prove Theorem 4.

Before we describe our algorithm, we provide some intuition about the general connection between the inclusion-exclusion principle and the Hamming distance between a pair of strings. Our key insight is that the inclusion-exclusion principle allows us to express whether two strings have Hamming distance bounded by, say k. The following lemma makes this idea precise:

▶ **Lemma 23** (Hamming Distance by Inclusion-Exclusion). *Let x and y be two strings of length d over some alphabet Σ, and let $0 \leq k < d$. Then:*

$$\mathbf{1}(\operatorname{HD}(x, y) \leq k) = \sum_{\substack{I \subseteq [d] \\ |I| \geq d-k}} (-1)^{|I|-d+k} \cdot \binom{|I| - 1}{d - k - 1} \cdot \mathbf{1}(x[I] = y[I]).$$

Recall that we write $x[I] = y[I]$ to express that the strings x and y are equally restricted to the indices in I. The precise inclusion-exclusion-type formula does not matter too much here, but we provide some intuition for Lemma 23 by considering the special cases where $\operatorname{HD}(x, y) = k$ and $\operatorname{HD}(x, y) = k - 1$. If $\operatorname{HD}(x, y) = k$, then there is a unique set I of size $d - k$ for which $x[I] = y[I]$. If instead $\operatorname{HD}(x, y) = k - 1$, then there is a unique such set of size $d - k + 1$, and additionally there are $d - k + 1$ such sets of size $d - k$. The scalars $(-1)^{|I|-d+k}\binom{|I|-1}{d-k-1}$ are chosen in such a way that in any case, all these contributions sum up to exactly 1.

The takeaway from the above lemma is that we can express the proposition that two strings satisfy $\operatorname{HD}(x, y) \leq k$ by a linear combination of 2^d indicators of the form $\mathbf{1}(x[I] = y[I])$. It is easy to extend this idea further to the following lemma, which is the core of our combinatorial approach:

▶ **Lemma 24** (Radius by Inclusion-Exclusion). *Let x be a string of length d over some alphabet Σ, let X be a set of strings each of length d over Σ, and let $0 \le k < d$. Then $r(x, X) \le k$ if and only if*

$$|X| = \sum_{\substack{I \subseteq [d] \\ |I| \ge d-k}} (-1)^{|I|-d+k} \cdot \binom{|I|-1}{d-k-1} \cdot |\{y \in X : x[I] = y[I]\}|.$$

Given this lemma, our algorithm for the Closest String problem is easy to state. Informally, we proceed in the following two steps:

Step 1: Partition. Precompute, for all $x \in X$ and for all $I \subseteq [d]$, the value $|\{y \in X : x[I] = y[I]\}|$. This can be implemented in time $O(n \cdot 2^d \cdot \mathrm{poly}(d))$ by partitioning the input strings X depending on their characters in the range I. After computing this partition, we can read the value $|\{y \in X : x[I] = y[I]\}|$ as the number of strings in the same part as x.

Step 2: Inclusion-Exclusion. We test for each $0 \le k \le d$ and $x \in X$, whether $r(x, X) \le k$ and finally return the best answer. By Lemma 24 we can equivalently express the event $r(x, X) \le k$ via

$$|X| = \sum_{\substack{I \subseteq [d] \\ |I| \ge d-k}} (-1)^{|I|-d+k} \cdot \binom{|I|-1}{d-k-1} \cdot |\{y \in X : x[I] = y[I]\}|.$$

By observing that the sum contains only 2^d terms and noting that we have precomputed the values $|\{y \in X : x[I] = y[I]\}|$, we can evaluate the sum, for a fixed x, in time $O(2^d \cdot \mathrm{poly}(d))$. In total, across all strings $x \in X$, the running time becomes $O(n \cdot 2^d \cdot \mathrm{poly}(d))$.

Finally, let us briefly comment on the $\mathrm{poly}(d)$ term in the running time. When evaluating the above sum naively, we naturally incur a running time overhead of $\mathrm{poly}(d)$ since the numbers in the sum need $\Omega(d + \log n)$ bits to be represented. However, this overhead can be circumvented by evaluating the expression in a smarter way. We provide more details in Section 4.1.

4.1 The Algorithm in Detail

In this subsection, we provide our algorithms for the discrete Closest String problem. Let us first demonstrate how to precompute $|\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ efficiently.

▶ **Lemma 25.** *We can compute $|\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ in time $O(n \cdot 2^d)$.*

Proof. Our strategy is to compute, for each $I \subseteq [d]$, a partition $\mathcal{P}_I$ of the set of all strings X such that two strings $y_1, y_2 \in X$ are in the same part in $\mathcal{P}_I$ if and only if $y_1[I] = y_2[I]$. This is our goal since, for all strings $x \in X$, the value we are interested in $|\{y \in X : x[I] = y[I]\}|$ is exactly the size of the part P in $\mathcal{P}_I$ that contains x. Thus, if we can efficiently compute, for all $I \subseteq [d]$ and all $x \in X$, the partition $\mathcal{P}_I$ and the part $P \in \mathcal{P}_I$ such that $x \in P$ then we have the desired algorithm.

Computing the partition $\mathcal{P}_I$ for each subset of $I \subseteq [d]$ when $|I| \le 1$ is simple: The partition $\mathcal{P}_\emptyset$ contains just one part which is the entire input set. We also know that $\mathcal{P}_{\{i\}} = \{\{x \in X : x[i] = \sigma\} : \sigma \in \Sigma\}$ for every $0 \le i \le d-1$. Thus, we can compute the partitions $\mathcal{P}_\emptyset$ and $\mathcal{P}_{\{i\}}$ for every $0 \le i \le d-1$ in time $O(n \cdot d)$. The remaining question is how to efficiently compute the partitions $\mathcal{P}_I$ for each subset of $I \subseteq [d]$ where $|I| \ge 2$.

The idea is to use dynamic programming in combination with a *partition refinement* data structure. Let us start with some notation: For a partition $\mathcal{P}$ and a set S, we define the *refinement* of $\mathcal{P}$ by S as the partition $\{P \cap S, P \setminus S : P \in \mathcal{P}\}$. For two partitions $\mathcal{P}$ and $\mathcal{P}'$, we define the refinement of $\mathcal{P}$ by $\mathcal{P}'$ by the iterative refinement of all sets $S \in \mathcal{P}'$. In previous work, Habib, Paul, and Viennot [16] have established a data structure to maintain partitions $\mathcal{P}$ of some universe $[n]$ that efficiently supports the following two operations:

- *Refinement:* We can refine a partition $\mathcal{P}$ by another partition $\mathcal{P}'$ in time $O(n)$.
- *Query:* Given a partition $\mathcal{P}$ and an element $i \in [d]$, we can find the part $i \in P \in \mathcal{P}$ in time $O(1)$.

Given this data structure, our algorithm is simple: Enumerate all sets I in nondecreasing order with respect to their sizes $|I|$. Writing $I = I' \cup \{i\}$ (for some $i \in [d]$), we compute $\mathcal{P}_I$ as the refinement of the previously computed partitions $\mathcal{P}_{I'}$ and $\mathcal{P}_{\{i\}}$. It is straightforward to verify that this algorithm is correct. The running time of each refinement step is $O(n)$ and so the total running time is $O(n \cdot 2^d)$ as claimed. ◀

We are finally ready to state our algorithm and prove its correctness using Lemmas 24 and 25.

Proof of Theorem 4. First, it is clear that if we test for each $0 \le k \le d$ and $x \in X$ whether $r(x, X) \le k$ then we can find the solution to the discrete Closest String problem. From Lemma 24 we know that $r(x, X) \le k$ if and only if:

$$|X| = \sum_{\substack{I \subseteq [d] \\ |I| \ge d-k}} (-1)^{|I|-d+k} \cdot \binom{|I|-1}{d-k-1} \cdot |\{y \in X : x[I] = y[I]\}|.$$

Thus, if we efficiently compute $|\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ and efficiently compute the right-hand side of the equation we will have an efficient algorithm for the discrete Closest String problem. We know from Lemma 25 that we can precompute $|\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ in time $O(n \cdot 2^d)$. Therefore, the only missing part of the algorithm is computing the inclusion-exclusion step in $O(n \cdot 2^d)$ time.

If we naively evaluate the inclusion-exclusion formula the running time becomes $\Omega(n \cdot 2^d \cdot d)$ as the intermediate values need $\Omega(d)$ bits to be represented in memory. However, we observe that inclusion-exclusion formula can indeed be evaluated more efficiently by rewriting it as follows:

$$\begin{aligned}
&\sum_{\substack{I \subseteq [d] \\ |I| \ge d-k}} (-1)^{|I|-d+k} \cdot \binom{|I|-1}{d-k-1} \cdot |\{y \in X : x[I] = y[I]\}| \\
&\quad = \sum_{\ell=d-k}^{d} (-1)^{\ell} \cdot \binom{\ell-1}{d-k-1} \cdot \sum_{\substack{I \subseteq [d] \\ |I| = \ell}} |\{y \in X : x[I] = y[I]\}|.
\end{aligned}$$

We can precompute $S[x, \ell] := \sum_{I \subseteq [d], |I| = \ell} |\{y \in X : x[I] = y[I]\}|$ for all strings $x \in X$ and all values $1 \le \ell \le d$ before we compute the inclusion exclusion step. Since there are 2^d different subsets of $[d]$ and since we already have access to the values $|\{y \in X : x[I] = y[I]\}|$, for all strings $x \in X$, computing $S[x, \ell]$ amounts to time $O(n \cdot 2^d)$. Afterwards, computing

$$\sum_{\ell=d-k}^{d} (-1)^{\ell} \cdot \binom{\ell-1}{d-k-1} \cdot S[x, \ell]$$

for all strings $x \in X$ and for all $0 \le k \le d-1$ only takes time $O(n \cdot d^3)$. Hence, the total running time of the algorithm is $O(n \cdot 2^d)$. ◀

Algorithm 1 An algorithm for the discrete Closest String problem in the small-distance regime. See Theorem 4.

(Step 1: Precompute $T[x, I] = |\{y \in X : x[I] = y[I]\}|$*)*
$\mathcal{P}_\emptyset \leftarrow X$
$\mathcal{P}_{\{i\}} \leftarrow \{\{x \in X : x[i] = \sigma\} : \sigma \in \Sigma\}\ \forall i \in [0, \dots, d-1]$
for $I = I' \cup \{i\}$ **do**
    $\mathcal{P}_I \leftarrow$ refinement of $\mathcal{P}_{I'}, \mathcal{P}_{\{i\}}$
for $x \in X, I \subseteq [d]$ **do**
    $T[x, I] \leftarrow |P|$ where $x \in P \in \mathcal{P}_I$

(Step 2: Inclusion-Exclusion)
for $x \in X, I \subseteq [d]$ **do**
    $S[x, |I|] \leftarrow S[x, |I|] + T[x, I]$
for $k \leftarrow 0, \dots, d-1$ **do**
    for $x \in X$ **do**
        if $|X| = \sum_{\ell=d-k}^{d} (-1)^{\ell} \cdot \binom{\ell-1}{d-k-1} \cdot S[x, \ell]$ **then**
            return x
return an arbitrary $x \in X$

We summarize the pseudocode of the algorithm outlined in the proof of Theorem 4 in Algorithm 1.

References

1 Amir Abboud, MohammadHossein Bateni, Vincent Cohen-Addad, Karthik C. S., and Saeed Seddighin. On complexity of 1-center in various metrics. *CoRR*, abs/2112.03222, 2021. arXiv:2112.03222.

2 Amir Abboud, Richard Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230. SIAM, 2015. doi:10.1137/1.9781611973730.17.

3 Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 377–391. SIAM, 2016. doi:10.1137/1.9781611974331.ch28.

4 Josh Alman, Timothy M. Chan, and R. Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 467–476. IEEE Computer Society, 2016. doi:10.1109/FOCS.2016.57.

5 Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 136–150. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.18.

6 Noga Alon, Rina Panigrahy, and Sergey Yekhanin. Deterministic approximation algorithms for the nearest codeword problem. In Irit Dinur, Klaus Jansen, Joseph Naor, and José D. P. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 12th International Workshop, APPROX 2009, and 13th International Workshop, RANDOM 2009, Berkeley, CA, USA, August 21-23, 2009. Proceedings*, volume 5687 of *Lecture Notes in Computer Science*, pages 339–351. Springer, 2009. doi:10.1007/978-3-642-03685-9_26.

7 Lijie Chen. On the hardness of approximate and exact (bichromatic) maximum inner product. *Theory Comput.*, 16:1–50, 2020. doi:10.4086/toc.2020.v016a004.

8 Gérard Cohen, Iiro Honkala, Simon Litsyn, and Antoine Lobstein. *Covering Codes*. ISSN. Elsevier Science, 1997.

9 Cláudio Nogueira de Meneses, Zhaosong Lu, Carlos A. S. Oliveira, and Panos M. Pardalos. Optimal solutions for the closest-string problem via integer programming. *INFORMS J. Comput.*, 16(4):419–429, 2004. doi:10.1287/ijoc.1040.0090.

10 Joaquín Dopazo, A. Rodriguez, J. C. Saiz, and Francisco Sobrino. Design of primers for PCR amplification of highly variable genomes. *Comput. Appl. Biosci.*, 9(2):123–125, 1993. doi:10.1093/bioinformatics/9.2.123.

11 Moti Frances and Ami Litman. On covering problems of codes. *Theory Comput. Syst.*, 30(2):113–119, 1997. doi:10.1007/s002240000044.

12 Leszek Gasieniec, Jesper Jansson, and Andrzej Lingas. Efficient approximation algorithms for the hamming center problem. In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 905–906. ACM/SIAM, 1999. URL: http://dl.acm.org/citation.cfm?id=314500.315081.

13 Jens Gramm, Falk Huner, and Rolf Niedermeier. Closest strings, primer design, and motif search. In *Sixth Annual International Conference on Computational Molecular Biology*, June 2002.

14 Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, 37(1):25–42, 2003. doi:10.1007/s00453-003-1028-3.

15 Venkatesan Guruswami, Daniele Micciancio, and Oded Regev. The complexity of the covering radius problem on lattices and codes. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 161–173. IEEE Computer Society, 2004. doi:10.1109/CCC.2004.1313831.

16 Michel Habib, Christophe Paul, and Laurent Viennot. A synthesis on partition refinement: A useful routine for strings, graphs, boolean matrices and automata. In Michel Morvan, Christoph Meinel, and Daniel Krob, editors, *STACS 98, 15th Annual Symposium on Theoretical Aspects of Computer Science, Paris, France, February 25-27, 1998, Proceedings*, volume 1373 of *Lecture Notes in Computer Science*, pages 25–38. Springer, 1998. doi:10.1007/BFb0028546.

17 Ishay Haviv and Oded Regev. Hardness of the covering radius problem on lattices. *Chic. J. Theor. Comput. Sci.*, 2012, 2012. URL: http://cjtcs.cs.uchicago.edu/articles/2012/4/contents.html.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. doi:10.1006/jcss.2000.1727.

19 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

20 Yuval Kochman, Arya Mazumdar, and Yury Polyanskiy. The adversarial joint source-channel problem. In *Proceedings of the 2012 IEEE International Symposium on Information Theory, ISIT 2012, Cambridge, MA, USA, July 1-6, 2012*, pages 2112–2116. IEEE, 2012. doi:10.1109/ISIT.2012.6283735.

21 J. Kevin Lanctôt. *Some String Problems in Computational Biology*. PhD thesis, University of Waterloo, 2004.

22 J. Kevin Lanctôt, Ming Li, Bin Ma, Shaojiu Wang, and Louxin Zhang. Distinguishing string selection problems. *Inf. Comput.*, 185(1):41–55, 2003. doi:10.1016/S0890-5401(03)00057-9.

23 Ming Li, Bin Ma, and Lusheng Wang. On the closest string and substring problems. *J. ACM*, 49(2):157–171, 2002. doi:10.1145/506147.506150.

24 K. Lucas, M. Busch, S. Mossinger, and J. A. Thompson. An improved microcomputer program for finding gene- or gene family-specific oligonucleotides suitable as primers for polymerase chain reactions or as probes. *Comput. Appl. Biosci.*, 7(4):525–529, 1991. doi:10.1093/bioinformatics/7.4.525.

25 Bin Ma and Xiaoming Sun. More efficient algorithms for closest string and substring problems. *SIAM J. Comput.*, 39(4):1432–1443, 2009. doi:10.1137/080739069.

26 Holger Mauch, Michael J. Melzer, and John S. Hu. Genetic algorithm approach for the closest string problem. In *2nd IEEE Computer Society Bioinformatics Conference, CSB 2003, Stanford, CA, USA, August 11-14, 2003*, pages 560–561. IEEE Computer Society, 2003. doi:10.1109/CSB.2003.1227407.

27 Arya Mazumdar, Yury Polyanskiy, and Barna Saha. On chebyshev radius of a set in hamming space and the closest string problem. In *Proceedings of the 2013 IEEE International Symposium on Information Theory, Istanbul, Turkey, July 7-12, 2013*, pages 1401–1405. IEEE, 2013. doi:10.1109/ISIT.2013.6620457.

28 Daniele Micciancio. Almost perfect lattices, the covering radius problem, and applications to ajtai's connection factor. *SIAM J. Comput.*, 34(1):118–169, 2004. doi:10.1137/S0097539703433511.

29 V. Proutski and Edward C. Holmes. Primer master: a new program for the design and analysis of PCR primers. *Comput. Appl. Biosci.*, 12(3):253–255, 1996. doi:10.1093/bioinformatics/12.3.253.

30 Noah Stephens-Davidowitz and Vinod Vaikuntanathan. Seth-hardness of coding problems. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 287–301. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00027.

31 Patrick Traxler. The time complexity of constraint satisfaction. In Martin Grohe and Rolf Niedermeier, editors, *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 190–201. Springer, 2008. doi:10.1007/978-3-540-79723-4_18.

32 Ying Wang, Wei Chen, Xu Li, and Bing Cheng. Degenerated primer design to amplify the heavy chain variable region from immunoglobulin cdna. *BMC Bioinform.*, 7(S-4), 2006. doi:10.1186/1471-2105-7-S4-S9.

33 Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1207–1215. SIAM, 2018. doi:10.1137/1.9781611975031.78.

34 Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005. doi:10.1145/1077464.1077466.