Recovery from Non-Decomposable Distance Oracles

Zhuangfei Hu*, Xinda Li*, David P. Woodruff†, Hongyang Zhang*, Shufan Zhang*
*University of Waterloo, †Carnegie Mellon University
*{zhuangfei.hu, xinda.li, hongyang.zhang, shufan.zhang}@uwaterloo.ca, †dwoodruf@cs.cmu.edu

***Abstract*—A line of work has looked at the problem of recovering an input from *distance queries*. In this setting, there is an unknown sequence $s \in \{0,1\}^{\leq n}$, and one chooses a set of queries $y \in \{0,1\}^{\mathcal{O}(n)}$ and receives $d(s,y)$ for a distance function d. The goal is to make as few queries as possible to recover s. Although this problem is well-studied for *decomposable* distances, i.e., distances of the form $d(s,y) = \sum_{i=1}^{n} f(s_i, y_i)$ for some function f, which includes the important cases of Hamming distance, ℓ_p-norms, and M-estimators, to the best of our knowledge this problem has not been studied for non-decomposable distances, for which there are important instances including edit distance, dynamic time warping (DTW), Fréchet distance, earth mover's distance, and others. We initiate the study and develop a general framework for such distances. Interestingly, for some distances such as DTW or Fréchet, exact recovery of the sequence s is provably impossible, and so we show by allowing the characters in y to be drawn from a slightly larger alphabet this then becomes possible. In a number of cases we obtain optimal or near-optimal query complexity. One motivation for understanding non-adaptivity is that the query sequence can be fixed and provide a non-linear embedding of the input, which can be used in downstream applications involving, e.g., neural networks for natural language processing.**

***Index Terms*—Sequence Recovery, Edit Distance, DTW Distance, Fréchet Distance.**

I. Introduction

We study the problem of exact recovery of a sequence from queries to a distance oracle. Suppose there is an unknown input sequence s with length at most n, defined on a binary alphabet $\{0,1\}$. Assume we have a distance oracle which returns the distance $d(s,q)$ between a query sequence q and the unknown sequence s, where the query sequence q is chosen either adaptively or non-adaptively. The problem is to determine the sequence s with a minimal number of queries to the distance oracle. This problem has been studied for decomposable distances, that is, the distance function between two sequences can be computed as the sum of distances between pairs of characters at the same entry, but never for non-decomposable distances. Among all non-decomposable distances, we are particularly interested in the edit distance, (p)-Dynamic Time Warping (p-DTW), and Fréchet distances. The edit distance measures the minimum number of edit operations (i.e., insertions, deletions, and substitutions) for transforming one sequence to another. The p-DTW distance ($1 \leq p < \infty$) between two sequences x, y is defined as the minimum ℓ_p distance between two equal-length expansions of x, y, where the expansion of a sequence means you can duplicate each character of each sequence an arbitrary number of times. When $p = 1$, the p-DTW distance is called the DTW distance. If we consider the ℓ_∞ norm instead of the ℓ_p norm, we obtain the Fréchet distance.

The problem of exact recovery for *decomposable distances* is well-studied in the literature, under the names of the coin-weighing problem [2], [3] and the group testing problems [4], [5], [6]. The coin-weighing problem is to identify the weight of each coin from a collection of n coins, each being of weight either w_0 or w_1 (w_0 and w_1 are distinct). In this problem, our only access to the coins is via weighing a subset of the coins on a spring scale. The group testing problem has also been shown to be equivalent to the coin-weighing problem in some settings [7]. This line of research has been extensively studied with interesting applications. For example, the coin-weighing problem can be found in the detection problem [8], the problem of determining a collection [9], and the distinguishing family problem [10].

The query complexity of the adaptive version of the problem is also related to the original Mastermind game [11]. The Mastermind problem can be phrased as guessing an input sequence based on Hamming distance queries. The non-adaptive version of this problem can be shown to be equivalent to the well-studied non-adaptive coin-weighing problem [3]. One can then consider other variants of the Mastermind game where the input sequence is guessed based on other distance metrics, such as permutation-based distances [12], ℓ_p distances [13] and graph distances [14], [15]. However, general distance metrics that do not decompose into coordinate sums are less understood. In this paper, we initiate the study of this exact recovery problem on *non-decomposable* distances.

One motivation of our exact recovery problem is its application to adversarially robust learning on discrete domains. It is well-known that deep neural networks are vulnerable to adversarial examples: test inputs that have been modified slightly in the ℓ_p space can lead to problematic machine predictions. Though there exist various techniques such as Pixel-DP [16] and randomized smoothing [17] that achieve certified robustness against ℓ_p-norm perturbations in continuous domains, in many tasks such as natural language processing, the ℓ_p norm is not well-defined for discrete perturbations. To resolve this issue, inputs from a discrete domain are usually mapped to vectors in the ℓ_p space before being passed to a classifier; this is also known as a word embedding. We require two properties of such a mapping: 1) zero information loss; 2) Lipschitzness with respect to the distance metric in the input space. We show that

The preliminary version of this paper has been accepted by ITCS 2023 [1]. Authors are in alphabetical order.

Zhuangfei Hu, Xinda Li, Hongyang Zhang, and Shufan Zhang are with the Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L3G1, Canada. (email: {zhuangfei.hu, xinda.li, hongyang.zhang, shufan.zhang}@uwaterloo.ca)

David P. Woodruff is with the Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA. (email: dwoodruf@cs.cmu.edu)

the exact recovery problem yields a direct construction of such mappings: suppose the set of query sequences is $\{q_1,\dots,q_m\}$ and s is the unknown input sequence; the mapping for s: $\phi(s)=[d(s,q_1),\dots,d(s,q_m)]$ has Lipschitz constant at most $\sqrt{m}$ (in the ℓ_2 norm) and maintains complete information about s. Similar to edit distance, which can be used for describing the adversarial capability in changing sequences, the DTW and Fréchet distances have received significant attention for their flexibility in handling temporal sequences. The special instance of our problem on DTW and Fréchet distances may be useful for analyzing the robustness of DTW neural networks [18].

A distance embedding further inspires theoretical applications in functional analysis [19]. While the space of input sequence s is a metric space, it may not be a Hilbert space with a definition of norm and inner product. Our result provides us with a tool to define a mapping from a metric space to a Hilbert space without loss of information about the input sequences. One can then use the norm or inner product to analyze input sequences, e.g., when two input sequences are orthogonal and how to normalize an input sequence to have norm 1.

A. Our Contribution and Results

To the best of our knowledge, this paper makes the first effort to consider the non-decomposable distance recovery problem. We first present a general framework to tackle with this problem, and then exhaustively explore representative distances of this class, i.e., edit distance, DTW distance, and Fréchet distance. We also study the role of adaptivity and non-adaptivity and obtain a number of results on lower bounds and upper bounds of query complexity. Before introducing our technical results, we would like to clarify the assumptions we make in the setting of the problem and justify some of them.

Assumptions. Throughout the paper, we assume the alphabet of the unknown input sequence s is {0,1}. We note that under this assumption, all of our results for DTW described below will apply to p-DTW. To recover the sequence s, we submit adaptive or non-adaptive query sequences to a distance oracle. As we will show in Section I-A1, for some distance metrics, there exist input sequences that cannot be distinguished by any sequence on a binary alphabet. Therefore, our query sequences may be allowed to utilize alphabets outside {0,1}with $\mathcal{O}(1)$ extra characters to exactly recover the input sequence. For edit distance, the extended alphabet can contain *any symbol* outside the binary alphabet, as the edit distance oracle counts the edit operations no matter what symbol is used. For (p-)DTW distance and Fréchet distance, the extended alphabet can consist of *any real number*. We assume the maximum length of s is n, while the exact length of s is unknown.

Extension to non-binary inputs. The binary input sequence setting is not an over-simplified assumption. All the results we obtain on the binary setting can be naturally extended to any non-binary alphabet Σ by encoding the non-binary alphabet in a binary domain. This will increase the query complexity by a constant factor from $|\Sigma|$ (one-hot encoding) to $\log(|\Sigma|)$ (binary encoding). Though this may not be the best solution if one considers a large alphabet, this extension works for the results for all distance metrics shown in this paper. Improvement on this extension to the recovery problem leaves room for future research.

Optimality. Throughout the paper, we consider asymptotic optimality, that is, the asymptotic complexity lower and upper bounds match orderwise. We would like to investigate lower bounds of the problem per distance instance, and develop algorithms that shows upper bounds can match lower bounds up to constant factor or logarithmic factor (under Big-O / Big-Omega tilde notation).

To list the results we obtain on this non-decomposable distance recovery problem, we begin with a general coordinate descent framework that can help recover sequences from a large class of distance oracles, including but not limited to earth mover's distance (EMD), cascaded norms (ℓ_p of ℓ_q), and A norms (a.k.a. Mahalanobis distance). We then present improved results on three specific distance metrics: edit distance, DTW distance, and Fréchet distance. We first provide several observations on the sequence recovery problem, showing the existence of indistinguishable input sequences despite the fact that we can query their DTW and Fréchet distances with all possible binary query sequences. We also prove lower bounds on the query complexity in our distance recovery problem w.r.t. DTW, edit, and Fréchet distances. Then we present our main results on recovering sequences from edit, DTW, and Fréchet distance oracles, with adaptive and non-adaptive strategies.

1) Existence of Indistinguishable Sequences: We observe that, for some distances, there exist sequences that cannot be distinguished by any query sequence over a binary alphabet. This can be proved by showing concrete examples, i.e., a pair of sequences that cannot be distinguished, which we show is true for the DTW and the Fréchet distances, as stated in the following theorem.

Theorem I.1 (Informal, existence of indistinguishable sequences)**.** *There exists a pair of sequences (s,s') such that s and s' cannot be distinguished by any query sequence on a binary alphabet, for the DTW distance and the Fréchet distance.*

The formal proof of this theorem for the DTW distance is deferred to Theorem VI.1. The analogous discussion for the Fréchet distance can be found in Section VII. Due to the existence of indistinguishable sequences, we define the concept of an *equivalence class* of sequences, which is a set of input sequences which are indistinguishable from all queries by a given distance oracle.

This observation suggests the scope of the distance recovery problem we study. We further *categorize the recovery guarantee into the following three levels*, from strong to weak: 1) recover the **exact input sequence**; 2) recover any sequence in the **same equivalent class of the input sequence**, where the equivalence class is defined to be the set for which any two input sequences in the equivalence class cannot be distinguished by calling the distance oracle to all query sequences; 3) recover any sequence which has **zero distance to the input sequence**. While the third level is the weakest one, in certain cases it can be reduced to the first two levels—for norm-induced distance functions, the recovered sequence is exactly the input sequence; for semi-norm-induced distance functions, the recovered sequence is in the same equivalence class. For other distance functions which are not *metric*, recovering a sequence with zero distance to input does not necessarily imply any one of the first two levels. We will show that our general coordinate descent framework can recover sequences with the third-level guarantee.

2) General Coordinate Descent *Framework for Adaptively Querying Distance Oracles:* We develop a general framework for recovering an input sequence from adaptive queries, which models the problem as a *zero-th order optimization* and utilizes a coordinate-descent-based algorithm to give a solution. The *coordinate descent* framework defines the distance between the input sequence and the query sequence as the loss function. The objective of the optimization is to reduce the loss function to 0, which guarantees what we call the *third level of recovery*. We define a *step operation* to modify the query sequence. For example, in the context of edit distance, a step operation is defined as adding/removing/substituting a character of the query sequence. To perform coordinate descent, our algorithm performs one step operation each time and queries the oracle to find a direction for which the loss decreases by at least a pre-determined constant scalar. By iteratively performing this method, the loss can be reduced to 0 and we show that the overall complexity of this method is $\mathsf{poly}(n)$, given that the maximum length of the sequence is n. For a large class of non-decomposable distance functions, such as the earth mover's distance (EMD), the cascaded norm (ℓ_p of ℓ_q), and the A norm, we can use this framework to yield a solution, as stated in the following theorem.

Theorem I.2 (Coordinate Descent for Adaptive Distance Queries)**.** *For an arbitrary distance oracle, a binary alphabet $\{0,1\}$ and any input sequence $s \in \{0,1\}^i$ where $0 \le i \le n$, using* coordinate descent *can* reduce the distance to *the input sequence s* to 0*, by adaptively querying the distance oracle between s and a set of query sequences with query complexity at most* $\mathsf{poly}(n)$*.*

Sufficient conditions for using this framework and further details can be found in Theorem IV.1.

3) Lower Bounds on the Recovery Problem: If we study the problem of exact recovery (the first level of recovery), we can obtain an information-theoretic lower bound of $\tilde{\Omega}(n)$ for various distance oracles, given by the following theorem. Here $f(n) = \tilde{\Omega}(g(n))$ if $f(n) = \Omega(g(n)/\mathsf{polylog}(n))$.

Theorem I.3 (Lower Bounds for Exact Recovery)**.** *For any input sequence $s \in \{0,1\}^i$ where $0 \le i \le n$, if for any input sequence and query the distance oracle has* $\mathsf{poly}(n)$ *possible values, any algorithm which* exactly recovers *s by querying the distance oracle between s and a set of query sequences requires query complexity at least $\tilde{\Omega}(n)$.*

The idea behind this bound is that, there are exponentially many possible input sequences with length at most n, while for the distance oracles given in our setting, the output of each query is a distance between two sequences which only has $\mathsf{poly}(n)$ possibilities. Hence, we need at least $\log_{\mathsf{poly}(n)}(2^{n+1}) = \tilde{\Omega}(n)$ queries. We instantiate this theorem on the edit distance and DTW distance in Theorem V.1 and Theorem VI.11, for recovery to the exact input distance.

We note for the DTW distance and Fréchet distance, there exist indistinguishable sequences, which lead to the recovery problem for equivalence class. Since the total number of equivalence classes is less than the number of input sequences, the previous counting technique (based on simple facts from information theory) no longer works. So we need a different argument, as we give in the following theorem:

Theorem I.4 (Lower Bounds for Equivalence Class Recovery)**.** *For a binary alphabet $\{0,1\}$ and any input sequence $s \in \{0,1\}^i$ where $0 \le i \le n$, any algorithm which recovers the sequence s up to equivalence by querying the DTW or Fréchet distance oracle between s and a set of query sequences requires query complexity at least $\Omega(n)$.*

We highlight our techniques used in proving this lower bound in Section III, while the formal proof can be seen in Theorem VI.4 and Theorem VII.1.

4) Adaptively Querying Distance Oracles, Optimally: We first answer the distance recovery problem with adaptive query strategies. Our solutions are summarized in the theorem below.

Theorem I.5 (Upper Bounds for Adaptive Exact Recovery)**.** *For a binary alphabet $\{0,1\}$ and any input sequence $s \in \{0,1\}^i$ where $0 \le i \le n$, there exists an algorithm which can* exactly recover *the input sequence s, by* adaptively *querying the distance oracle (for the* edit *and* DTW *distances) between s and a set of query sequences with query complexity at most $\mathcal{O}(n)$.*

All results in Theorem I.5 match our lower bounds on the query complexity. Without extra character(s), using the DTW distance oracle we can only recover a sequence in the same equivalence class. Our result in Theorem I.5 for the DTW distance is achieved with the assistance of 1 extra character outside the alphabet $\{0,1\}$, and the proof and algorithm can be found in Theorem IV.6.

For the edit distance, we have two different adaptive algorithms that can achieve the $\mathcal{O}(n)$ bound. The first algorithm makes use of the property that, for two sequences, the edit distance is equal to the difference in their lengths, if and only if one sequence is a subsequence of the other. We construct an $\mathcal{O}(n)$ adaptive query set and a binary search algorithm utilizing this property to recover the input sequence. Our second algorithm instead queries the length of the input sequence by an empty sequence and then finds a set of $\mathcal{O}(n)$ bases as the query set, from which we can reconstruct the input sequence. These are further detailed in Theorem IV.2 and Theorem IV.4.

For the Fréchet distance, adaptive and non-adaptive strategies are essentially the same, because we prove that $2n-1$ queries are necessary and sufficient for recovering from a Fréchet distance oracle. However, we can only recover a sequence in the equivalence class in this setting. This result is described as a non-adaptive query strategy in Theorem VII.3.

TABLE I
SUMMARY OF OUR RESULTS FOR RECOVERING ARBITRARY INPUT SEQUENCES OF LENGTH n UNDER THE CONSTRAINT THAT THE QUERY LENGTH IS OF $\mathcal{O}(n)$. LB: LOWER BOUND. #EC: NUMBER OF EXTRA CHARACTERS.

Oracle	Query Complexity	LB	Adaptive?	#EC	Level of Recovery	Positions
Edit	$2k\log(n/k)+k+\log n+c$ or $n+2$	$\tilde{\Omega}(n)$	Adaptive	0	Exact sequence	Theorems IV.2&IV.4
Edit	$n+1$	$\tilde{\Omega}(n)$	Non-adaptive	1	Exact sequence	Theorem V.2
Edit	$\frac{1}{2}(n^2+3n)$	$\tilde{\Omega}(n)$	Non-adaptive	0	Exact sequence	Theorem V.5
(p-)DTW	$n+1$	$\tilde{\Omega}(n)$	Adaptive	1	Exact sequence	Theorem IV.6
(p-)DTW	$2n$	$\Omega(n)$	Non-adaptive	0	Equivalent class	Theorem VI.7
(p-)DTW	n^2+n	$\tilde{\Omega}(n)$	Non-adaptive	1	Exact sequence	Theorem VI.12
(p-)DTW	$n+2$	$\tilde{\Omega}(n)$	Non-adaptive	2^{*}	Exact sequence	Theorem VI.14
Fréchet	$2n-1$	$2n-1$	N/A†	0^{**}	Equivalent class	Theorem VII.3
Any distance	$\mathsf{poly}(n)$	-	Adaptive	0	Zero distance to input	Theorem IV.1

† For both adaptively and non-adaptively querying the Fréchet distance oracle, the optimal bound on the query complexity is $2n-1$.

* Increasing #EC from 2 to an arbitrary constant cannot improve the query complexity to be better than $\tilde{\mathcal{O}}(n)$.

** Involving extra characters not only cannot improve the level of recovery from "equivalence class" to "exact sequence", but also cannot improve the query complexity (see Theorem VII.2).

5) Non-adaptively Querying Distance Oracles, Optimally: Next we describe our non-adaptive query strategies for the distance recovery problem. Theorem I.6 shows upper bounds for exact sequence recovery, while Theorem I.7 summarizes our results on the recovery problem of finding a sequence in the same equivalence class as the input sequence.

Theorem I.6 (Upper Bounds for Non-adaptive Exact Recovery)**.** *For a binary alphabet $\{0,1\}$ and any input sequence $s \in \{0,1\}^i$ where $0 \leq i \leq n$, there exists an algorithm which can* exactly recover *the input sequence s, by querying the distance oracle (for the* edit *and* DTW *distances) between s and a* non-adaptive *set of query sequences with query complexity at most $\mathcal{O}(n)$, with the assistance of $\mathcal{O}(1)$ extra characters in the query sequences.*

With 1 extra character, we show the construction of a set of non-adaptive queries that can exactly recover sequences from the edit distance (Theorem V.2), while with 2 extra characters, we can exactly recover input sequences from the DTW distance (Theorem VI.14). Both results match our lower bound on the query complexity, while we complement our results with an $\mathcal{O}(n^2)$ query complexity algorithm for the DTW distance with 1 extra character (Theorem VI.12). We note that non-adaptive strategies have limited power compared to adaptive strategies. Hence, we consider adding extra characters to construct query strategies that are comparable to the lower bounds. For the edit distance, introducing more than 1 extra characters cannot encode more information in the query results, because the cost between 0 (or 1) and any other additional character is always the same.

Theorem I.7 (Upper Bounds for Non-adaptive Equivalence Class Recovery)**.** *For a binary alphabet $\{0,1\}$ and any input sequence $s \in \{0,1\}^i$ where $0 \leq i \leq n$, there exists an algorithm which can* recover the sequence in the same equivalence class as *the input sequence s, by querying the distance oracle (for the* DTW *and* Fréchet *distances) between s and a* non-adaptive *set of query sequences with query complexity at most $\mathcal{O}(n)$, without extra characters in the query sequence.*

By Theorem I.7, if we are not allowed to use extra characters, we can only recover the sequence in the same equivalence class as the input sequence for the DTW distance. Our query construction and proof are shown in Theorem VI.7. We also remark that for Fréchet distance, using extra characters cannot help to improve the results of Theorem VII.3, as shown in Theorem VII.2.

Summary. The main technical results of this paper are summarized in Table I.

B. Paper Roadmap

The remainder of the paper is organized as follows. Section II introduces the notations and essential background definitions (regarding sequence, distances, and matching properties) used in this paper. Section III highlights the techniques and insights behind our proofs of non-adaptively querying the DTW distance oracle, which helps the understanding of the most non-trivial and interesting parts of this paper. Section IV consists of our results on the recovery problem with adaptive queries, which begin with a general framework for all non-decomposable distances and follow by instantiations as per distance using specific properties. We present and discuss our results on the lower bounds and upper bounds of query complexity for recovery with non-adaptive queries on edit distance, DTW distance, and Fréchet distance, with different recovery guarantees, in Section V, Section VI and Section VII, respectively. Section VIII summarizes the related papers to our problem. As an initiation of this line of study in the recovery of non-decomposable distances, we finally describe the yet-open problems in Section IX.

II. PRELIMINARIES

We would like to briefly introduce the fundamental concepts, definitions and notations that are involved in this paper. An alphabet is a finite set of characters. A binary alphabet contains two elements, $\Sigma_b := \{0, 1\}$. A sequence is either empty ϕ, or an enumerated collection of characters selected from a given alphabet. We denote the length of a sequence s by $\mathrm{len}(s)$. Throughout the paper, we use $[n]$ to denote the set $\{1, 2, \ldots, n\}$. Then for sequence s, $[\mathrm{len}(s)]$ represents its indices set. Note our indices set starts from 1.

A distance function between a pair of sequences measures the similarity and the structural relationship between them. A distance function $\mathrm{dist}(\cdot, \cdot)$, as a *metric*, satisfies the following properties:

- *Identity*: $\mathrm{dist}(s, s') = 0$ iff $s = s'$;
- *Commutativity*: $\mathrm{dist}(s, s') = \mathrm{dist}(s', s)$;
- *Triangle inequality*: for any sequence x, $\mathrm{dist}(s, s') \leq \mathrm{dist}(s, x) + \mathrm{dist}(x, s')$;
- *Non-negativity*: $\mathrm{dist}(s, s') \geq 0$.

Different distance functions can capture the similarity information from different perspectives. While we use $\mathrm{dist}(\cdot, \cdot)$ to denote the distance metric in general, in this paper we are in particular interested in the edit distance (denoted by $d_L(\cdot, \cdot)$, L for Levenshtein), (p)-Dynamic Time Warping (p-DTW) distance (denoted by $d_{\mathrm{DTW}}(\cdot, \cdot)$), and Fréchet distance (denoted by $d_F(\cdot, \cdot)$), which are non-decomposable to a sum of coordinate-wise contributions. We note that the widely used DTW distance is *not a metric* because identity and triangle inequality properties do not hold for it. It has been shown a generalization to p-th power of DTW (i.e., p-DTW) distance satisfies weak triangle inequality up to a factor parameterized by p and the sequence length [20]. We discuss in this paper how the missing triangle inequality affects our recovery problem (especially for DTW).

There are several other definitions related to sequences that are useful in our paper.

Definition II.1 (Runs and Expansion, [21])**.** The runs of a sequence x are the maximal substrings consisting of a single repeated character. Any sequence obtained from x by extending x's runs is an expansion of x. For a given character c, we use c^m to represent the sequence obtained by repeating c for m times. We denote the length of the i-th run of x by $\textsc{lor}(x, i)$, where $\textsc{lor}$ means *Length of Run* function, and the number of runs of a sequence x by $\#\textsc{runs}(x)$.

The following definitions of a condensed expression and subsequence are useful in developing our algorithms.

Definition II.2 (Condensed Expression)**.** We say y is a condensed expression of x if (i) y has the same number of runs as x, (ii) the first and last character of y and x are the same, (iii) each run of y only has 1 character.

Definition II.3 (Subsequence and Substring)**.** Given a sequence y, its subsequence x is derived by deleting zero or more characters from y without changing the order of the remaining characters. The substring x' is a *contiguous subsequence* of y. We use $x[a]$ to denote the a-th character of the sequence x, and $x[a, b]$ to denote a substring of x which starts from the a-th character and ends at the b-th character.

As an example, consider the sequence 0010111. The number of runs in this sequence is 4. The runs of sequence 0010111 are 00 (the 1st run), 1 (the 2nd run), 0 (the 3rd run), and 111 (the last run), with length of $2, 1, 1, 3$, respectively. By duplicating the characters, we can extend a run in a sequence and then obtain another sequence which is an expansion of the original one. For instance, by extending the second run in 0010111, we get 0011110111 which is the expansion of sequence 0010111. The condensed expression of 0010111 is the sequence 0101. Sequences 010, 101, 0111 are subsequences (or substrings) of 0010111, while 01111, 000, 1111 are only subsequences (not substrings).

The definitions of these three distances (Edit, DTW, and Fréchet) are listed as follows.

Definition II.4 (Edit Distance, or Levenshtein Distance [22])**.** Given two sequences x and y, the edit distance $d_L(x, y)$ equals the *minimal number* of *edit operations* required for a sequence x to be transformed to sequence y. Specifically, we consider the Levenshtein distance [22] which captures the addition, deletion, and substitution of single symbols.

We use $\|\cdot\|_1$ or simply $\|\cdot\|$ to denote the ℓ_1 norm distance between two equi-length sequences whose symbols are real numbers. The notation for absolute value $|\cdot|$ is used to calculate the cost or difference between two characters.

Definition II.5 (DTW Distance, [21])**.** Consider two sequences x, y of length m_1 and m_2, respectively. A correspondence $(\overline{x}, \overline{y})$ between x and y is a pair of equal-length expansions of x and y. The cost of a correspondence is calculated as the ℓ_1 distance between $\overline{x}, \overline{y}$: $\|\overline{x} - \overline{y}\|_1$. A correspondence between x and y is said to be optimal if it has the minimum attainable cost, and the resulting cost is called the dynamic time warping distance $d_{\mathrm{DTW}}(x, y)$, that is $d_{\mathrm{DTW}}(x, y) = \min_{(\overline{x}, \overline{y}) \in \mathcal{W}_{x,y}} \|\overline{x} - \overline{y}\|_1$, where $\mathcal{W}_{x,y}$ denotes the set of all correspondences $(\overline{x}, \overline{y})$.

Definition II.6 (p-DTW Distance, [20])**.** By replacing the ℓ_1 norm in Definition II.5 with the ℓ_p norm ($1 \leq p < \infty$), we obtain the definition for the p-DTW distance.

In addition to the existing definitions, we need to introduce some new concepts essential to our proofs for (p)-DTW distance.

Definition II.7 (Monotonic Sequence)**.** Recall that the indices set of sequence x is denoted by $x.\mathrm{indices} := [\mathrm{len}(x)]$. We say a sequence x is monotonic, if for every $i, j \in [\mathrm{len}(x)]$, $i < j \Rightarrow x_i \leq x_j$, or for every $i, j \in [\mathrm{len}(x)]$, $i < j \Rightarrow x_i \geq x_j$, where x_i denotes the i-th character in x.

Definition II.8 (Matching)**.** Consider the query sequence q and the input sequence s as two *vertex sets* ($U = \{u_1, \ldots, u_\ell\}, V = \{v_1, \ldots, v_n\}$) where the vertex set U denotes the characters in sequence q and the vertex set V denotes the characters in sequence s. Let M be an *edge set* that for each $m := (u, v) \in M$, we have $u \in U$ and $v \in V$. We say $M(q, s)$ is a *matching* (or simply M when the context is clear) between q and s (or U and V) if M satisfies the following properties:

1) every vertex in U and V corresponds to at least one edge in M;

2) the first character in U is matched to the first character in

TABLE II
SUMMARY OF MAIN NOTATIONS

Notation	Meaning of Notation	Notation	Meaning of Notation
s	The input sequence	ϕ	The empty sequence
$s[i]$	The i-th character of sequence s	$s[i,j]$	A substring of s (from the i-th to the j-th character)
c^m	Repeating character c for m times	$\text{len}(s)$	The length of s
$[n]$	$\{1,2,\ldots,n\}$	$[\text{len}(s)]$	The index set of s
$\text{LOR}(s,i)$	The length of the i-th run of s	$\#\text{RUNS}(s)$	The number of runs in s
$\mathcal{Q}$	Query set	$q^{(i)}$	The i-th query in the query set
$\text{dist}(\cdot,\cdot)$	The general distance oracle	$d_L(\cdot,\cdot)$	The edit distance oracle
$d_{\text{DTW}}(\cdot,\cdot)$	The DTW distance oracle	$d_F(\cdot,\cdot)$	The Fréchet distance oracle
$\text{MSS}(seq,r)$	A MSS instance	$\|\cdot\|_p$	ℓ_p norm

V and the last character in U is matched to the last character in V;

3) the indices of matched character pairs are monotonic, i.e., for any two edges $(u_i,v_j),(u_k,v_l)\in M$, $i>k\Rightarrow j\geq l$ and $j>l\Rightarrow i\geq k$.

We define the degree of a vertex, $\deg(u_i)$ or $\deg(v_j)$, as the number of associated edges in a matching M.

Definition II.9 (DTW Matching)**.** The cost of an edge $m:=(u_i,v_j)\in M$ is defined to be the ℓ_1 norm distance $\text{Cost}(m):=\|u_i-v_j\|$. The cost of a matching is defined as $\text{Cost}(M):=\sum_{m\in M}\text{Cost}(m)$. Let $\mathcal{M}$ consist of all possible matchings between q and s (or U and V). If a matching $M\in\mathcal{M}$ has *minimal cost* on the edges, that is $\text{Cost}(M)=\min_{M_i\in\mathcal{M}}\text{Cost}(M_i)$, we call this matching a *DTW matching*. A *DTW matching* yields a *DTW distance* between q and s.

Based on our definitions, the concepts of matching provide a different perspective of the non-decomposable distance. A matching between two vertex sets defines a possible alignment between two sequences with different lengths. The notion of DTW matching better captures the graph-theoretical properties of the implicit optimal alignment in computing DTW distance than the conventional definition. The cost of a DTW matching is equal to the DTW distance between two sequences which are constituted by the vertex sets respectively. We note that there might exist multiple DTW matchings (of equal cost) between a pair of sequences.

Definition II.10 (Isomorphic Matching)**.** Given input sequence s of length ℓ, two query sequences q and q' of length n and two corresponding matchings M (between q and s) and M' (between q and s'). We say M and M' are *isomorphic* if, $\forall 1\leq i\leq\ell$ and $\forall 1\leq j\leq n$, edge $(s_i,q_j)\in M \iff$ edge $(s_i,q'_j)\in M'$.

Definition II.11 (Fréchet Distance)**.** By replacing the ℓ_1 norm in Definition II.5 with the ℓ_∞ norm, we obtain the definition of the Fréchet distance.

The Fréchet distance in our paper is equivalent to the discrete Fréchet distance in the prior works of [23], [24].

Extended alphabet. Since in this paper we discuss recovery sequence based on distance queries from *binary or extended alphabet*, we would like to note that the distance definitions are independent of the alphabets. That being said, while we study the problem by restricting the input sequence as drawn from the binary alphabet (which generalizes to any constant-sized alphabet by applying coding methods), we do not change the distance definitions in a skewed way of embedding special symbols on the extended alphabet or backdoors to the oracle. To ensure that the distance output makes sense, we specify the extended alphabets for queries to the different distance oracles. For edit distance, the extended alphabet can contain *any symbol* outside the binary alphabet, as the edit distance oracle counts the edit operations no matter what symbol is used. For (p-)DTW distance and Fréchet distance, the extended alphabet can consist of *any real number*. This makes sense because the DTW and Fréchet distances are defined based on ℓ_p or ℓ_∞ cost.

The main notations used in this paper are summarized in Table II.

III. OUR TECHNIQUES

In this section, we summarize and highlight the main technical insights behind our results on non-adaptive recovery from the DTW distance oracle, which are the most non-trivial and interesting parts of this paper. We hope to convey our intuitive ideas in a less formal manner before diving into the full proofs in the later sections. Reader may skip this section if they are looking for the complete statements and proofs of these results. In particular, we will cover the intuitions behind the following four theorems.

Theorem III.1 (Hardness, Refers to Theorem VI.1)**.** *There exists a pair of input sequences s and s' such that for any query sequence q, $d_{\text{DTW}}(s,q)=d_{\text{DTW}}(s',q)$. That is, s and s' cannot be distinguished by DTW Distance Oracle queries without using extra characters.*

Theorem III.1 shows the impossibility of only using binary sequences to recover the input sequence from the DTW distance oracle. If two input sequences cannot be distinguished, we say that they are in the same *equivalence class*. The following two informal theorems state the upper bound and lower bound on DTW distance recovery up to the equivalence class.

Theorem III.2 (Informal, Upper Bound, Refers to Theorem VI.7)**.** *There exists a query set $\mathcal{Q}$ consisting of $\mathcal{O}(n)$ queries of length $\mathcal{O}(n)$, such that any two distinguishable input sequences can be distinguished by $\mathcal{Q}$.*

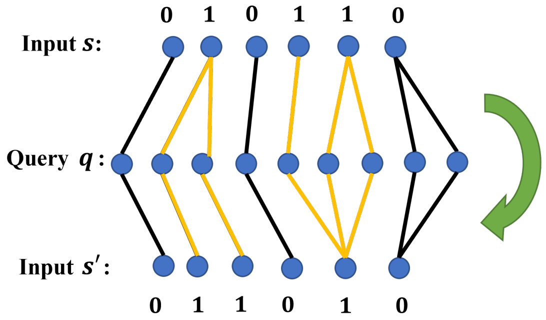

Fig. 1. Constructing a matching between q and s' based on the matching between q and s.

$\mathcal{Q}$ is designed to contain all queries with i runs, for any $i \in [1, n]$.

Theorem III.3 (Informal, Lower Bound, Refers to Theorem VI.4)**.** *For the binary alphabet $\{0, 1\}$, any algorithm to recover an arbitrary input sequence $s \in \{0, 1\}^\ell$, where $0 \leq \ell \leq n$, up to its equivalence class, by querying the DTW distance to a set of sequences, has query complexity $\Omega(n)$.*

Note that our upper bound matches the lower bound for DTW equivalence class recovery. The next exciting finding is that, using queries that contain a small number of extra characters, we can exactly recover the input sequence.

Theorem III.4 (Informal, Upper Bound with Extra Chars, Refers to Theorem VI.14)**.** *By introducing $\mathcal{O}(1)$ extra characters to the query sequence alphabet, we can recover any input sequence of length $\leq n$ with $\mathcal{O}(n)$ DTW queries.*

We aim to recover the given input sequence (of length $\leq n$) with the minimum number of queries for different distance metrics. Theorems III.1, III.2 and III.3 summarize the best results one can hope to obtain for recovering sequences from a DTW oracle without extra characters, i.e., identifying the equivalence class that the input sequence belongs to. If we are allowed to use extra characters in the query construction, we can distinguish and recover all the sequences with $\mathcal{O}(n)$ queries, as informally stated in Theorem III.4. We summarize and highlight the techniques used in proving these theorems in the rest of this section, in which the informal proofs are grouped as follows. In Section III-A, we show proof sketches on recovery of sequences using binary queries, which include results from Theorems III.1, III.2 and III.3. In Section III-B, we give a bird's-eye view over the key ideas of the query construction and proof of Theorem III.4.

A. Optimal Non-adaptive Strategy using DTW Queries over Binary Alphabet

The hardness result (Theorem III.1) is shown by finding evidence of such a pair of indistinguishable input sequences.

Informal proof for Theorem III.1. In the case of DTW Distance, we discover that it is actually impossible to recover any given input with an arbitrary number of queries. For example, the input sequences $s = 010110$ and $s' = 011010$ cannot be exactly recovered, since they cannot be distinguished by any query sequence. To see this, the idea is that $d_{\mathrm{DTW}}(1, r) = d_{\mathrm{DTW}}(11, r)$ for any non-empty sequence r, unless $r = 0$. Therefore, a DTW matching between s and any query sequence q would yield a corresponding matching between s' and q with the same cost, (see Figure 1 as an example) and vice versa. (Refer to Theorem VI.1 for detailed proof). This implies that $d_{\mathrm{DTW}}(s, q) = d_{\mathrm{DTW}}(s', q)$, and thus s and s' cannot be distinguished by q.

Before giving the intuition for the proof of Theorem III.2 and III.3, we first introduce the notion of a Min 1-Seperated Sum (MSS) problem [25], [26], where each instance of the DTW distance computation can be reduced to solving a corresponding instance of MSS problem. The reduction plays the role of an important primitive in our proofs.

MSS Problem. The min 1-separated sum (MSS) problem takes as input a sequence seq of m positive integers and an integer $r \geq 0$. The problem is to select r integers from seq and minimize their sum, under the constraint that any two adjacent integers cannot be selected simultaneously. We say $\mathrm{MSS}(seq, r)$ is an MSS instance.

Theorem III.5 (DTW-to-MSS Reduction, [26], Theorem 2)**.** *Let $x \in \{0, 1\}^m$ and $y \in \{0, 1\}^n$ be two binary strings such that $x[1] = y[1], x[m] = y[n]$, and $\#\textsc{runs}(x) \geq \#\textsc{runs}(y)$. Then, the DTW distance between x and y, i.e., $d_{\mathrm{DTW}}(x, y)$, equals the sum of a solution for the MSS instance* $\mathrm{MSS}\left(\left(\textsc{lor}(x, 2), \ldots, \textsc{lor}(x, \#\textsc{runs}(x) - 1)\right), \frac{(\#\textsc{runs}(x) - \#\textsc{runs}(y))}{2}\right)$.

To give an example of the reduction, let $s = 010110$ and $q = 010$. By Theorem III.5, we obtain $d_{\mathrm{DTW}}(s, q) = \mathrm{MSS}((1, 1, 2), 1)$. For ease of presentation, we will use $\mathrm{MSS}(x, (\#\textsc{runs}(x) - \#\textsc{runs}(y))/2)$ to represent the same MSS instance.

Remark. For binary strings $x \in \{0, 1\}^m, y \in \{0, 1\}^n$ where $x[1] \neq y[1]$ or $x[m] \neq y[n]$, we can still reduce $d_{\mathrm{DTW}}(x, y)$ to an MSS instance (which will be presented later in the paper using another technique from [26]). In this section, where we only illustrate the main idea of the proofs, we will only consider the case where the input sequence and query sequence each have the same starting character and the same ending character (so Theorem III.5 can be directly applied), and other cases can be resolved similarly. For full details, we defer to later sections.

Intuition for Theorem III.2. We would like to skip the proof sketch for Theorem III.2, but just to mention the insights of the query construction to obtain such an orderwise optimal query complexity upper bound. The set of queries $\mathcal{Q}$ contains queries of all possible combinations of runs in the input sequence. That is, for the maximum length n of the input sequence, the set of the possible number of runs is $[n]$. This gives us n queries. Since we have 0 runs and 1 runs, there are $2n$ queries in the query set $\mathcal{Q}$ in total. Then the remainder of the proof is to perform case analysis – we first eliminate obvious cases and then build a mapping to the corresponding MSS instances such

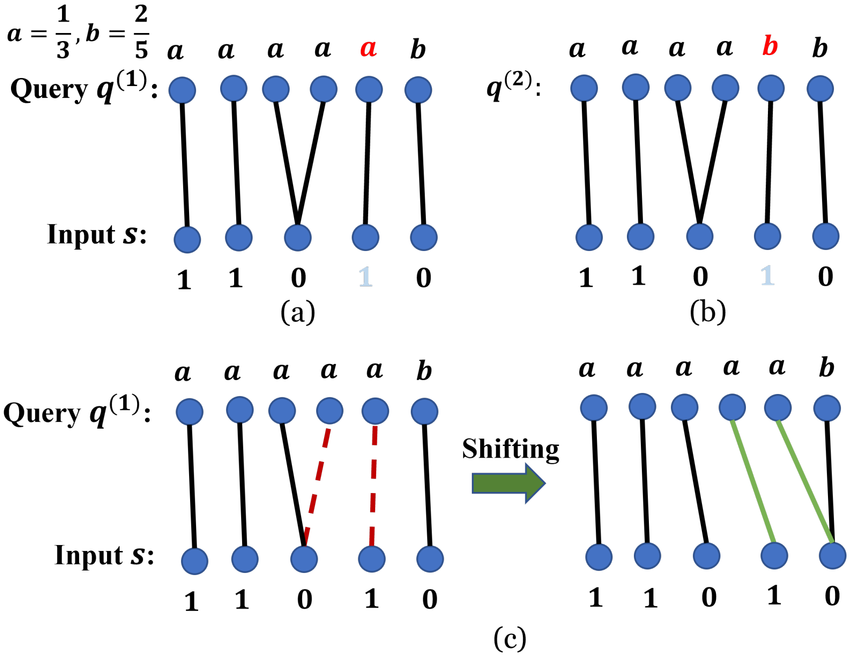

Fig. 2. (a) Illustration of input-uniqueness and 0/1-uniqueness; (b) Illustration of isomorphism and performing a difference operation, compared to Fig (a); (c) Illustration of shifting operation.

that if any pair of sequences cannot be distinguished by $\mathcal{Q}$, they cannot be distinguished by any binary queries.

Informal proof of Theorem III.3. Recall our query set $\mathcal{Q}$ contains queries of all numbers of runs. The intuition for the proof of Theorem III.3 is that, for each given constant-length interval of the number of runs, we can construct a certain pair of input sequences which can only be distinguished by queries with a number of runs within this interval. For instance, it can be proved that $s_1 = 01^3 01^3 0^3 1^3 0^3 1^3 0$ and $s_2 = 01^3 0^2 1^3 0^2 1^3 0^3 1^3 0$ can only be distinguished with queries with a number of runs within $[4, 10]$. Thus, an $\Omega(n)$ number of such constructed pairs of input sequences can correspond to $\Omega(n)$ disjoint intervals, yielding an $\Omega(n)$ lower bound for this problem.

We now construct a class of pairs of input sequences (s, s') where s and s' share the same starting and ending character, such that s and s' can only be distinguished by queries q with a number of runs within $[\#\text{RUNS}(\text{s})+c_1, \#\text{RUNS}(\text{s})+c_2]$ for two constants $c_1 < c_2$. According to Theorem III.5, as long as the constructed pair of input sequences (s, s') have the same number of runs, for a query q with more than $\#\text{RUNS}(s)$ number of runs, $d_{\text{DTW}}(q, s)$ and $d_{\text{DTW}}(q, s')$ are only determined by the query q and $\#\text{RUNS}(s)$, and thus q cannot distinguish s and s'. For a query q with fewer than $\#\text{RUNS}(s)$ number of runs, $d_{\text{DTW}}(q, s)$ and $d_{\text{DTW}}(q, s')$ are reduced to two MSS instances. Note that for different queries q, the sequences (i.e., the first parameter) of MSS instances remain the same, while $\#\text{RUNS}(q)$ determines the number of elements selected in the sequences of MSS instances (i.e., $(\#\text{RUNS}(s) - \#\text{RUNS}(q))/2$). We hope to construct a pair of sequences seq and seq' such that $\text{MSS}(seq, 1) \neq \text{MSS}(seq', 1)$ and $\text{MSS}(seq, x) = \text{MSS}(seq', x)$ for all $x > 1$: let seq and seq' be the sequences corresponding to MSS instances of s and s'; in this way, s and s' would still be distinguishable because $\text{MSS}(seq, x) \neq \text{MSS}(seq', x)$ for $x = 1$, but any query q with fewer than $\#\text{RUNS}(s) - 4$ runs cannot distinguish s and s' because $\text{MSS}(seq, x) = \text{MSS}(seq', x)$ for all $x \geq 2$, where $x = (\#\text{RUNS}(s) - \#\text{RUNS}(q))/2$.

B. Optimal Non-adaptive Strategy using DTW Queries with Extra Characters

We show that, if we augment the ability of our oracles by introducing extra characters, we can solve the DTW distance oracle recovery problem with optimal query complexity up to polylogarithmic factors.

Informal proof of Theorem III.4. We would like to construct a query set of size $\mathcal{O}(n)$ that can recover the input sequence using a DTW distance oracle. A natural idea is to retrieve information about the input sequence by taking the difference between the query results of neighbouring queries (i.e., queries only differing by 1 character). To achieve this, we construct a query set satisfying the following three properties:

1) *Isomorphism*: The matchings corresponding to neighboring queries should be isomorphic. Fig. 2 (a) and Fig. 2 (b) show an example of isomorphism, where only one character of the input sequence is changed, while the structure of both optimal matchings remains identical. With this property, we know that the difference between the query results of neighboring queries only reflects the effects of the different characters in neighboring queries. This property is the essence of guaranteeing the correctness of the difference operation.

2) *Input-uniqueness*: Each character in the query sequence should be matched to exactly 1 character in the input sequence.

Another way to think of this property is to imagine a *total function* that maps the entire query sequence to the input sequence. Each matching between the query and input defines such a function so that we can extract information about the input by knowing something about the function. With this property, we can take the difference to get the information of a single character in the input sequence with a pair of neighboring queries. Note that if the differing character in the neighboring queries is matched to multiple characters in the input sequence, the difference in the query results can only reflect the sum of the costs over these characters, which makes exact recovery hard. Take Fig. 2 (a) and Fig. 2 (b) as an example. Input-uniqueness is satisfied for both Fig. 2 (a) and Fig. 2 (b), since all characters in the query sequences of both figures have degree 1. Denote the matchings from Fig. 2 (a) and Fig. 2 (b) by M_a and M_b respectively. Since M_a has cost $3(1-a)+2a+b$ while M_b has cost $2(1-a)+2a+(1-b)+b$, we know that $\mathrm{Cost}(M_a)-\mathrm{Cost}(M_b)=b-a$. By taking the difference, we can infer that $s[4]=1$; otherwise, if $s[4]=0$, we would have $\mathrm{Cost}(M_a)-\mathrm{Cost}(M_b)=(a-0)-(b-0)=a-b$.

Combining properties 1) and 2), we note that each character in the input sequence can match to 1 or more characters in the query sequence, so we can obtain an expansion of the input sequence. Based on the example, Fig. 2 (a) and Fig. 2 (b), we can obtain an expansion, 110010, of the input sequence. We can then infer that the input sequence is of the form 1^x0^y10, where $x,y\in[1,2]$. To recover the exact input sequence, we require more information given by the following third property.

3) *0/1-uniqueness*: In an optimal matching w.r.t. our constructed queries, either all 0's or all 1's in the input sequence have degree 1. Using this property, we can locate the exact position of either all 0's or all 1's in the input sequence, and exactly recover the input sequence by combining the two cases. In the example of Fig 2, 1-uniqueness is satisfied in Fig. 2 (a) and Fig. 2 (b), while 0-uniqueness is not, since $s[3]$ in both figures has degree 2. According to 1-uniqueness, we can reduce the form of the input sequence from 1^x0^y10 to 110^y10. Similarly, we can construct another set of queries that satisfies 0-uniqueness to locate the positions of 0's in the input sequence, which determines y in this example.

Sequence Monotonicity → Input-uniqueness. We observe that property 2) can be obtained from a *monotonic* design of the query sequences.

Lemma III.6 (Refers to Lemma VI.15)**.** *Given a monotonic sequence q of length n where*

$$\min_{i\in[n]}\max\{|q[i]-0|,|q[i]-1|\} > \max_{i,j\in[n]}|q[i]-q[j]|, \qquad (1)$$

for any input sequence s with length $\ell\le n$, given a DTW matching M for (q,s), we have $\deg(q[i])=1$ for all characters $q[i]$ in q.

The intuition for Lemma III.6 is that, with the monotonic property and equation (1) guaranteed in our query construction, we can ensure that there do not exist characters $s[i]\in s$ and $q[j]\in q$ where $\deg(s[i])>1$ and $\deg(q[j])>1$ are satisfied at the same time. Fig 4 in a later section illustrates that, for such a pair of $s[i]$ and $q[j]$, we can always construct a matching with lower cost where one of their degrees is decreased to 1. Therefore, either all characters in s or all characters in q would have degree 1. Since $\mathrm{len}(q)=n\ge\mathrm{len}(s)$, we know that $\deg(q[i])=1$ for all characters $q[i]$ in q.

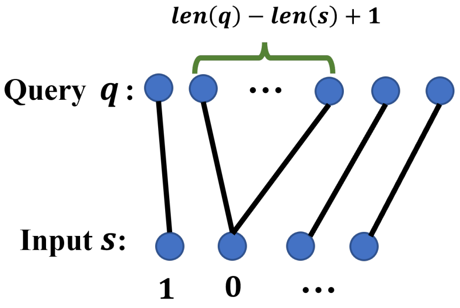

Fig. 3. The position of the first 0 in s and the length of both sequences can determine the structure of M_i.

Fig. 2 (a) and Fig. 2 (b) satisfy sequence monotonicity, since the query sequences in both figures are monotonic sequences of length n and for x in $\{1,2\}$, $\min_{i\in[n]}\max\{|q^{(x)}[i]-0|,|q^{(x)}[i]-1|\}=\frac{3}{5}>(\frac{2}{5}-\frac{1}{3})=\max_{i,j\in[n]}|q^{(x)}[i]-q^{(x)}[j]|$.

Sequence 0/1-preference → 0/1-uniqueness. We observe that property 3) can be guaranteed by the *0/1-preferred* design of the query sequences. If all characters in the query sequence are less than (or greater than) $\frac{1}{2}$, then we can guarantee 1-uniqueness (or 0-uniqueness) of the input sequence. Intuitively, this would hold because, if all characters in the query sequence are less than (or greater than) $\frac{1}{2}$, matching them to 0's (or 1's) in the input sequence yields lower cost than matching to 1's (or 0's). Fig. 2 (a) and Fig. 2 (b) satisfy 0-preference, since all characters in query sequences (either $a=\frac{1}{3}$ or $b=\frac{2}{5}$) are less than $\frac{1}{2}$.

Query Construction. We now propose the following design of the query sequence. We first need a single 0 query and a single 1 query to obtain the number of 1's and 0's in the input sequence. Let a,b be two fractional characters that satisfy $0<b-a<a<b<\frac{1}{2}$ and the denominators of a,b are co-prime. Without loss of generality, we can assume $a=\frac{1}{3}$ and $b=\frac{2}{5}$. We will use a,b as the extra characters to construct the query sequences. In particular, the rest of the query sequences (other than the 0 query and the 1 query) consist of queries $\mathcal{Q}$ in the form of $q^{(i)}=a^{n-i}b^i$, where $i=1,\ldots,n$. This query construction satisfies *sequence monotonicity* and *sequence 0/1-preference* properties. Now we need to prove it also satisfies *isomorphism.*

Lemma III.7 (Refers to Lemma VI.19)**.** *For any input sequence s, there exists an set of isomorphic matchings $\mathcal{M}$ where $M_i\in\mathcal{M}$ is optimal for query $q^{(i)}\in\mathcal{Q}$.*

Lemma III.7 guarantees the isomorphism property of the constructed query set $\mathcal{Q}$. Here we construct an isomorphic set of matchings $M_i\in\mathcal{M}$ such that only the first 0 in the input sequence has degree greater than 1, while all other characters in the matching are of degree 1. Fig. 2 (a) and Fig. 2 (b) are instances of M_1 and M_2, where the matchings in both figures are isomorphic to each other. Note that in this construction, the structure of the matchings is only determined by the position of the first 0 in the input sequence and the length of both sequences (see Fig 3). Since all query sequences in $\mathcal{Q}$ have the same length, an isomorphism of constructed matchings is naturally guaranteed.

To prove the optimality of the M_i, we introduce the notion of a *"shifting" operation*. Consider two 0's in the input sequence. If any character between them has degree 1 and the first 0 has degree greater than 1, by running the shifting operation we decrease the degree of the first 0 by 1 and increase the degree of the last 0 by 1, while preserving the degree of all other characters. Fig. 2 (c) illustrates an example of the shifting operation.

Claim III.8 (Refers to Claim VI.20)**.** For our constructed query set $\mathcal{Q}$, a shifting operation would not reduce the total cost of the matching.

Claim III.9 (Informal, Refers to Claim VI.21)**.** Given input sequence s, query $q^{(i)} \in \mathcal{Q}$ and any optimal matching M_i^* between s and $q^{(i)}$, we can obtain M_i^* by applying a series of shifting operations to M_i.

Combining the above two claims, we can show that the M_i are always optimal, which proves Lemma III.7. So far, the constructed query set satisfies three properties – isomorphism, input-uniqueness, and 0/1-uniqueness. Further details of our algorithm to recover the input sequence are given in later sections (see Algorithm 2).

IV. Recovery with Adaptive Queries

A. General Framework

Theorem IV.1 (Coordinate Descent Framework)**.** *For a given distance oracle* $\operatorname{dist}(\cdot,\cdot)$*, a constant-sized alphabet* Σ *and any input sequence* $s \in \Sigma^i$ *where* $0 \leq i \leq n$*, there exists an adaptive algorithm which returns a sequence* s' *such that its distance to the input sequence* s *satisfies* $\operatorname{dist}(s,s') = 0$ *using* $\mathsf{poly}(n)$ *queries, given that the following two conditions are true:*

- *There exists a positive constant c (independent of n),* $\forall s \in \Sigma^i, q \in \Sigma^{\mathcal{O}(n)}$*, where* $0 \leq i \leq n$ *and* $\operatorname{dist}(s,q) > 0$*, we can find a sequence* q' *within* $\mathsf{poly}(n)$ *queries such that* $\operatorname{dist}(s,q) \geq \operatorname{dist}(s,q') + c$*;*
- $\forall s \in \Sigma^i, q \in \Sigma^{\mathcal{O}(n)}$, $\operatorname{dist}(s,q) \leq \mathsf{poly}(n)$*.*

Proof sketch: The two above conditions naturally imply a local search algorithm. To recover the sequence q, we perform the following steps: 1) randomly initialize q. 2) find q' such that $\operatorname{dist}(s,q) > \operatorname{dist}(s,q')$. 3) set q to q' and repeat 2) to 3). The algorithm terminates if $\operatorname{dist}(s,q) = 0$, and outputs the final q as the sequence s'.

Since we reduce $\operatorname{dist}(s,q)$ by at least a positive constant c in each iteration, and $\operatorname{dist}(s,q) \leq \mathsf{poly}(n)$, the algorithm terminates in at most $\mathsf{poly}(n)/c$ iterations. Therefore, the total number of queries is $\mathcal{O}(\mathsf{poly}(n))$. □

The above local search algorithm can be applied to all aforementioned distances. Specifically, the complexity for the edit distance, DTW distance and Fréchet distance is $\mathcal{O}(n^2)$, $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$, respectively. A detailed instantiation of the algorithm on these distances can be found in Appendix B.

Remark. As stated in the theorem, the objective of this coordinate descent framework is to reduce $\operatorname{dist}(s,s')$ to 0, which reflects our "zero distance to input" recovery guarantee. We remark that, for distance function which is a *metric*, this guarantee implies "recover to equivalence class", while for distances such as DTW where the triangle inequality does not apply, there exist sequences that can be distinguished whereas the distance is 0. Such an example includes sequence 101 and 1011.

B. Edit Distance

We show that a binary input sequence with maximum length n can be adaptively recovered using at most $n + \log n + c \in \mathcal{O}(n)$ queries to the edit distance oracle (where c is a constant), by the following theorem.

Theorem IV.2 (Adaptive Strategy for Edit Distance)**.** *For a binary alphabet* $\{0,1\}$*, and any input sequence* $s \in \{0,1\}^\ell$ *with k runs where* $0 \leq \ell \leq n$*, there exists an adaptive algorithm to recover the input sequence* s *using at most* $2k\log(n/k) + \log n + k + c$ *queries* $\mathcal{Q}$ *of length* $\leq n$ *and the exact Levenshtein distance of* s *to each query sequence* $q_i \in \mathcal{Q}$*, where the query sequences use no extra characters.*

Proof. The proof makes use of the following claim.

Claim IV.3. Given two sequences x and y, the edit distance $d_L(x,y) = |\operatorname{len}(x) - \operatorname{len}(y)|$ if and only if x is a subsequence of y or y is a subsequence of x.

Proof of claim. Without loss of generality, we can assume that $\operatorname{len}(x) \geq \operatorname{len}(y)$. Since each insertion, deletion or substitution operation can change the sequence length by at most 1, we have $d_L(x,y) \geq t$ where $(t = \operatorname{len}(x) - \operatorname{len}(y))$. If y is a subsequence of x, we can obtain y by performing t deletions on x. Since $d_L(x,y) \geq t$, we have $d_L(x,y) = t$. If y is not a subsequence of x, we show that $d_L(x,y) > t$. To transform x to y we would need at least t deletions. Since y is not a subsequence of x, we cannot obtain y by merely performing t deletions on x, implying that $d_L(x,y) > t$.

Next, to prove Theorem IV.2, we observe that for any sequence on a binary alphabet, the first run starts with either 0 or 1. That is, the condensed expression of a binary sequence is in the form of 1010... or 0101.... Let the number of runs be k. The first part of our adaptive recovery algorithm is determining the input sequence's condensed expression. To do so, we need the following set of $2n+1$ queries, $\{\phi, 0, 1, 01, 10, 010, \dots\}$, where the maximum length of the query in this set is n. The length of the input sequence ℓ can be determined by querying the empty sequence ϕ. The condensed expression of the input sequence is equal to the query in the query set of maximum length such that $k = \ell - r$, where k is the length of this query and r is the query result from the oracle. Since we are adaptively querying the oracle, we do not require all $2n+1$ queries. By using our querying strategy, the query complexity of this part can be reduced to $\log n + c$. To see this, we take out all n queries beginning with 0 from the query set and adaptively query the oracle using binary search to find the longest query sequence such that the edit distance between this query and the input sequence equals the length difference between two sequences. Next, we add an 1 to the left side (or the most significant bit) of the longest query we just selected, then query the oracle to see if the distance is smaller. The query sequence with the smaller edit distance is therefore the condensed expression of the input sequence. Since a ϕ query is required at the beginning, the entire process requires at most $\log n + 2 \leq \log n + c$ queries.

The second part of our algorithm is to recover the sequence from the condensed expression via expanding each run by inserting 1's (or 0's) into the corresponding location. We have obtained the number of runs, which is k. According to the claim, if any one of the runs of the query sequence contains more characters than that of the input sequence, meaning the query sequence is no longer the subsequence of the input sequence, then we can observe from the query results. Therefore, we can recover the input sequence run by run. The naïve way of achieving this is to iterate over runs and insert one character per time to a run until scanning and fulfilling the entire sequence, which results in at most n queries. Combining the first part of recovering the condensed expression, this approach gives us the overall query complexity of $n + \log n + c \in \mathcal{O}(n)$.

An alternative approach to recover the runs is to determine the number of characters in each run using line search and binary search. That is, we increase the number of characters in a run exponentially (by a factor of 2) and then look back to find the exact number by binary search. Compared to directly using binary search to find the length of the run within the range of $[2, n-k]$, the complexity analysis of our approach can avoid a potential $n \log n$ term. To give an example of this approach, suppose we have a run of length 13. To recover this run, instead of using 13 queries by the naïve approach, we can make 7 queries with the following numbers of 1's: 2, 4, 8, 16, 12, 14, and 13, respectively. For a run with length m, the worst-case query complexity of this approach is $\lceil 2 \log m \rceil$. Let t_i be the number of characters in each 1 run or 0 run. Then we have $\sum_{i=1}^{k} t_i = \ell \leq n$. Since the number of characters in each block can be determined adaptively using line and binary search, we can derive the query complexity of the second part as $\sum_{i=1}^{k} \lceil 2 \log t_i \rceil \leq 2 \log(\Pi_{i=1}^{k} t_i) + k \leq 2 \log(\sum_{i=1}^{k} t_i / k)^k + k \leq 2k \log(n/k) + k$. The first inequality holds due to the AM-GM inequality. Combining the two parts of the algorithm, we know that the overall adaptive query complexity for the exact recovery of the sequence is $2k \log(n/k) + k + \log n + c \in \mathcal{O}(n)$. □

In many cases our alternative approach, as shown in the complexity, saves queries. There are also edge cases that the naïve approach wins the game – for runs with two characters, using binary search requires 3 queries (i.e., queries with 2, 4, 3, characters in this run respectively), while the naïve approach finishes the task with only 2 queries.

Theorem IV.4 (Yet Another Adaptive Strategy for Edit Distance)**.** *For a binary alphabet $\{0,1\}$, and any input sequence $s \in \{0,1\}^{\ell}$ where $0 \leq \ell \leq n$, there exists an adaptive algorithm to recover the input sequence s using at most $n + 2 \in \mathcal{O}(n)$ queries $\mathcal{Q}$ of length $\leq n$ and the exact Levenshtein distance of s to each query sequence $q_i \in \mathcal{Q}$, where the query sequences use no extra characters.*

Proof. The adaptive query strategy is the following. We first use an empty sequence to query the length $\ell \in [n] = \{1, 2, \ldots, n\}$ of the input sequence. Then we use $\ell + 1 \leq n + 1$ queries: an $e_0 = 0^{\ell}$ query and a set of $e_i = 0^{i-1}10^{\ell-i}, i \in [\ell]$ queries (all with length ℓ).

Claim IV.5. $s[i] = \begin{cases} 0, & \text{if } \; d_L(s, e_0) - d_L(s, e_i) \leq 0; \\ 1, & \text{if } \; d_L(s, e_0) - d_L(s, e_i) = 1. \end{cases}$

Proof of claim. If $s[i] = 1$, $d_L(s, e_0) - d_L(s, e_i) =$ (#1's in s) $-$ (#1's in $s - 1$) $= 1$. If $s[i] = 0$, $d_L(s, e_0) =$ (#1's in s). We show that $d_L(s, e_i) \geq$ (#1's in s). First, $d_L(s, e_i) \geq$ (#1's in s) $-$ (#1's in e_i) = #1's in s $-$ 1. Consider the series of transformations from s to e_i: 1) If we only perform substitution on s, we need at least #1's in s + 1 operations. 2) Otherwise we show that we have at least one insertion. If we perform at least one deletion operation(s) on s, since s and e_i are of the same length, we would need at least one insertion(s) on s. Note that insertions on s cannot reduce the difference of the number of 1's between s and e_i. Thus, we need at least (#1's in s -1) extra operations to reduce the difference to 0 and we have $d_L(s, e_i) \geq$ (#1's in s -1) $+1$ = #1's in s. Combining these cases, we obtain $d_L(s, e_i) \geq$ (#1's in s).

By claim IV.5, we can recover the sequence s character by character. □

Remark. We remark that both results of Theorem IV.2 and Theorem IV.4 are useful. Clearly, using $n + 2$ queries in the second algorithm is a better strategy than the naïve approach in the first algorithm, which requires $n + \log n + c$ queries. However, the first algorithm with the binary search approach yields query complexity of $2k \log(n/k) + \log n + 2k + c$. When the number of runs (i.e., k) is small, this result is better than the $n + 2$ queries in the second algorithm.

C. DTW Distance

Theorem IV.6 (Adaptive Strategy for DTW Distance)**.** *For a binary alphabet $\{0,1\}$, and any input sequence $s \in \{0,1\}^{\ell}$ where $0 \leq \ell \leq n$, there exists an adaptive algorithm to recover the input sequence s using at most $n + 1 \in \mathcal{O}(n)$ queries $\mathcal{Q}$ of length $\leq n$ and the exact DTW distance of s to each query sequence $q^{(i)} \in \mathcal{Q}$, where the query sequences use 1 extra character.*

Proof. Using an adaptive method, for a binary alphabet $\{0,1\}$, an input sequence $s \in \{0,1\}^i$ where $0 \leq i \leq n$ can be exactly recovered with at most $n + 1 \in \mathcal{O}(n)$ queries to the DTW distance oracle. We need 1 additional character, which is the fractional character $\frac{1}{2}$, to construct the set of query sequences. The details are presented as follows.

First, with a single-character query sequence $q^{(1)} = \frac{1}{2}$, we can obtain the length of the input sequence s, which is $\ell = 2d_{\mathrm{DTW}}(s, q^{(1)})$.

Consider the query $q^{(2)} = 0\ (\frac{1}{2})^{\ell-1}$. Note that each $\frac{1}{2}$ in the $q^{(2)}$ corresponds to at least $\frac{1}{2}$ cost in the query result, and we have $d_{\text{DTW}}(s, q^{(2)}) \geq (\ell-1)/2$. If $s[1] = 0$, then $s[1]$ and $q^{(2)}[1]$ are perfectly matched, so $d_{\text{DTW}}(s, q^{(2)}) = (\ell-1)/2$. Otherwise, the first character 0 in $q^{(2)}$ would correspond to cost > 0 in the query result, so $d_{\text{DTW}}(s, q^{(2)}) > (\ell-1)/2$. In this way, we can recover $s[1]$.

Now we recover the whole sequence by induction. Suppose we have recovered $s[1,k]$, we show that we can recover $s[k+1]$ with the query sequence $q^{(k+2)} = s[1,k]s[k](\frac{1}{2})^{\ell-k-1}$. Noting that each $\frac{1}{2}$ in $q^{(k+2)}$ corresponds to at least a $\frac{1}{2}$ cost in the query result, we have $d_{\text{DTW}}(s, q^{(k+2)}) \geq (\ell-k-1)/2$. If $s[k+1] = s[k]$, then $s[1,k+1]$ and $q^{(k+2)}[1,k+1]$ can be perfectly matched, so $d_{\text{DTW}}(s, q^{(k+2)}) = (\ell-k-1)/2$. Otherwise, we claim that $d_{\text{DTW}}(s, q^{(k+2)}) > (\ell-k-1)/2$. If the cost corresponding to $q^{(k+2)}[k+1] > 0$, we would already have $d_{\text{DTW}}(s, q^{(k+2)}) > (\ell-k-1)/2$, so we can assume that the cost corresponding to $q^{(k+2)}[k+1]$ is 0. Since $q^{(k+2)}[k+1] = s[k] \neq s[k+1]$, we know that $q^{(k+2)}[k+1]$ cannot be matched with $s[k+1]$. Suppose $q^{(k+2)}[k+1]$ is matched with substring $s[u, u+t]$ in the optimal DTW matching, where $t \geq 0$ and $s[u] = s[u+1] = \cdots = s[u+t] = q^{(k+2)}[k+1] = s[k]$. Since $k+1 \notin [u, u+t]$, we either have $k+1 > u+t$ or $k+1 < u$. If $k+1 > u+t$, since $\forall u+t < j \leq \ell$ we have $s[j]$ matched to a $\frac{1}{2}$, the total cost would be at least $(\ell-(u+t))/2 > (\ell-k-1)/2$. Otherwise if $k+1 < u$, note that $s[1,u]$ are matched to $q^{(k+2)}[1,k+1]$ in the optimal DTW matching. Since $s[k+1] \neq s[k]$, the number of runs in $s[1,k+1]$ would be greater than the number of runs in $q^{(k+2)}[1,k+1]$ by 1. Thus, the number of runs in $s[1,u]$ would be greater than the number of runs in $q^{(k+2)}[1,k+1]$ by at least 1, and they cannot be perfectly matched. Therefore, the cost corresponding to $q^{(k+2)}[1,k+1]$ would be greater than 0, yielding a total cost of greater than $(\ell-k-1)/2$.

By induction, we can recover the input sequence of maximum length n with $n+1$ queries. □

V. Recovery with Non-Adaptive Edit Distance Oracle Queries

We begin with a lower bound for edit distance.

Theorem V.1. *For a binary alphabet $\{0,1\}$, any algorithm to recover an arbitrary input sequence $s \in \{0,1\}^{\ell}$ where $0 \leq \ell \leq n$ by querying the Levenshtein distance to a set of sequences of length $\mathcal{O}(n)$ requires a query complexity of $\Omega(n/\log n)$.*

Proof. For each query of length $\mathcal{O}(n)$, the result would be an integer $d = \mathcal{O}(n)$. Without loss of generality, assume the query is of length $an+b$ where a, b are non-negative constant integers and $b < n$. For an arbitrary input sequence with length $\leq n$, the query result falls into the range of $[(a-1)n+b, an+b]$ if $a > 0$ (or $[0,n]$ if $a = 0$), yielding $n+1$ possibilities. For $0 \leq k \leq n$, the number of different sequences of length k is 2^k, and the total number of sequences of length no greater than n would be $\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$. Thus, to distinguish all possible sequences, one would need at least $\log_{n+1}(2^{n+1}) = (n+1)/\log(n+1) \in \Omega(n/\log n)$ queries.

Note that this information theoretical proof only applies to deterministic algorithms. Next we give a proof if one is allowed to use a randomized algorithm. To show this, we introduce a one-way two-party communication game called *INDEX*.

Definition V.1 (INDEX Game [27])**.** Consider two players Alice and Bob. Alice and Bob have access to a common public coin and their computation can depend on this. Alice holds an n-bit string $x \in \{0,1\}^n$ and is allowed to send a single message M to Bob (i.e., this is a one-way protocol). Bob has an index $i \in [n]$ and his goal is to learn $x[i]$, i.e., $\Pr_r[\mathcal{O}ut(M) = x[i]] \geq \frac{2}{3}$).

It is shown in [27] that the above problem requires $|M| = \Omega(n)$. To reduce our recovery problem from the INDEX game, let R be an adaptive randomized recovery algorithm which works as follows. First, Alice randomly selects a query q_1 based on the first part r_1 of the shared public coin, and computes $d(q_1, x)$. Alice then adaptively selects a set of queries q_i, where each q_i is chosen based on disjoint parts $r_1, \ldots, r_i$ of the public coin, as well as the responses $d(q_1, x), \ldots, d(q_{i-1}, x)$ to previous queries. Alice then sends all query results $d(q_i, x)$ to Bob as the message M.

We now show that, if the algorithm R is correct w.p. $2/3$, then M contains $\Omega(n/\log n)$ query results. Given the success probability $2/3$ of R, from message M, Bob can reconstruct the string x w.p. at least $2/3$, so Bob can learn each bit of x w.p. at least $2/3$. According to [27], $|M| = \Omega(n)$ bits. Since each distance query result contains at most $\mathcal{O}(\log n)$ bits, it follows that $\Omega(n/\log n)$ queries are required. □

A. Exact Recovery with Extra Character(s)

We now move on to the analysis of the upper bound for edit distance with the assistance of extra character(s). The following theorem uses 1 extra character in the extended alphabet to construct query sequences. We note that for edit distance, using more than 1 extra character in the extended alphabet does not help recover the input sequence, because the edit distance oracle only counts the edit operations made from transforming one sequence to another. Different characters result in the same edit cost.

Theorem V.2 (Non-adaptive Strategy for Edit Exact Recovery with 1 Extra Character)**.** *For a binary alphabet $\{0,1\}$ and an input sequence $s \in \{0,1\}^{\ell}$ where $0 \leq \ell \leq n$, there exists an algorithm to recover the input sequence s, given a set of $n+1 \in \mathcal{O}(n)$ query sequences $\mathcal{Q}$ of length $\leq n$ and the exact Levenshtein distance of s to each query sequence $q \in \mathcal{Q}$, where an extra character 2 is allowed in the query sequences.*

Proof of Theorem V.2. The intuition of our proof is to build an oracle that returns the number of 1's in the first j characters of the input sequence s. Then querying the oracle with all possible j's (where $j \in [n]$) implies a recovery of the input sequence. Note that this oracle calls the edit distance oracle as a subroutine. The following lemma shows the existence of such an oracle.

Lemma V.3. *Let $s \in \{0,1\}^{n'}$ be a non-empty sequence with length $\operatorname{len}(s) = n' \leq n$. Consider a sequence $s' = 1^j 2^{n-j}$,* *where $j \in [n']$, and* 2 *denotes a random character not in the binary alphabet* $\{0,1\}$*. Let k denote the number of* 1*'s in the substring $s[1,j]$ (i.e., the first j characters of s). Then the edit distance between s and s' is equal to $n-k$.*

Proof. We prove the lemma in two steps. First, we prove that the number of operations required in the transformation from s to s' is greater than or equal to $n-k$. Second, we show the existence of a sequence of operations that transforms s to s' in exactly $n-k$ steps.

To formally prove the first step, we perform a case analysis on the j-th character of s, i.e., $s[j]$, being 0 or 1. When $s[j]=0$, the following claim shows $d_L(s,s') \geq n-k$.

Claim V.4. If $s[j]=0$, then $d_L(s,s') \geq n-k$.

Proof of claim. Let $s = [\text{prefix}]0[\text{suffix}]$, where the length of the prefix is $\mathrm{len}([\text{prefix}]) = j-1$. Recall sequence $s' = 1^j2^{n-j}$. The edit distance between s and s' can be regarded as the number of operations required in the transformation from s to s'. This transformation from s to s' leads to a *sequence* of operations of insertion, deletion, and substitution. For an optimal transformation sequence, swapping two adjacent operations in this sequence generates another valid sequence of operations of the same length.We can therefore assume all the deletion operations are performed in the beginning, and we denote the number of deletions by d. Let t be the number of 0's in [prefix], and suppose $d \geq t+1$. We can assume that all entries in [suffix] are 1 and any 0 is deleted before a 1 is deleted; indeed, these assumptions will not increase the edit distance. After the deletion operations, s is a sequence of $n'-d$ 1's. If $n'-d \leq j$, then we need an additional $n-(n'-d)$ insertions to recover s'. Thus the total cost is $d+n-(n'-d) \geq d+n-j \geq (t+1)+n-j = n-k$. It remains to consider the case that $d < t+1$. At this point s is a sequence of length $n'-d$ containing $(t+1)-d$ 0's among its first j entries, and remaining 1's. Since there are no more deletions, any 1's occurring after the j-th entry must be substituted to a 2. There are $(n'-d)-j$ such 1's that each cost 1. Also, each of the $(t+1)-d$ 0's among the first j entries costs one for a substitution. Finally, we need at least $n-n'+d$ insertions to obtain equal-length sequences. So the total cost is at least $\underbrace{(d)}_{\text{\#deletions}} + \underbrace{(n-n'+d)}_{\text{\#insertions}} + \underbrace{[(n'-d)-j]}_{\text{\#substitutions of 1's}} + \underbrace{[(t+1)-d]}_{\text{\#substitutions of 0's}} = n-j+t+1 = n-k$. This completes all cases.

To finish the case analysis, now we consider the case that $s[j]=1$. We define $s'' = (s[1,j-1])0(s[j+1,n'])$. Note s'' is obtained by substituting the j-th character of s from 1 to 0, hence $d_L(s,s'')=1$. We have k 1's in $s[1,j]$, so we have $(k-1)$ 1's in $s''[1,j]$. By Claim V.4, we have $d_L(s',s'') \geq n-k+1$. Since edit distance is *a metric*, by triangle inequality, $d_L(s',s)+d_L(s,s'') \geq d_L(s',s'') \geq n-k+1$. Therefore, $d_L(s,s') \geq n-k+1-1 = n-k$.

Now, for the second step, we give a valid sequence of operations to transform s to s' in exact $n-k$ steps. 1) insert 2's to the end of s such that s and s' have the same length. This results in $n-n'$ insertions. 2) for every index $i \in [n]$, substitute $s[i]$ to $s'[i]$ if they are different in the first place. The number of operations is counted as follows. For $i \in [1,j]$, it requires $j-k$ substitutions since there are $(j-k)$ 0's. For $i \in [j+1,n']$, it requires $n'-j$ substitutions since we need to substitute every character to 2. For $i \in [n'+1,n]$, it requires no substitutions. Therefore, we have $n-n'+j-k+n'-j = n-k$ operations in total. □

Query Sequence Construction. We introduce an additional wildcard character which is not in the input sequence alphabet. Using this newly introduced character, the $n+1$ query sequences are constructed as follows. We use an empty sequence together with n sequences of the form of 1^j2^{n-j} for $j=1,\ldots,n$ where 2 denotes a "not-in-the-alphabet" character.

Algorithm to recover input sequence s. We now give an algorithm to recover s using the query sequence set to complete the proof of Theorem V.2. From the query result of the empty sequence, we know the length of the input sequence (n'). If $n'=0$, we know the input sequence is empty as well. Otherwise, consider the query results of the sequences 1^j2^{n-j}, for $j \in [n']$. By Lemma V.3, we know the number of 1's (k) in the first j characters of s ($j \in [n']$), which implies a complete recovery of s. The exact recovery algorithm is presented in Algorithm 1. We note that the order of the query sequences matters. ■

Remark. Note that the exact length of sequence s is unknown. This algorithm works non-adaptively for any sequence s with length $\leq n$.

B. Exact Recovery without Extra Characters

Theorem V.5. *For a binary alphabet $\{0,1\}$ and an input sequence $s \in \{0,1\}^\ell$ where $0 \leq \ell \leq n$, there exists an algorithm to recover the input sequence s, given $\frac{1}{2}(n^2+3n) \in \mathcal{O}(n^2)$ query sequences $\mathcal{Q}$ of length $\leq n$ and the exact Levenshtein distance of s to each query sequence $q_i \in \mathcal{Q}$, without extra characters.*

Proof. The construction can be obtained by naturally extending the query set in the proof of Theorem IV.4 to all lengths $\ell \in [n]$. This gives us $\sum_{\ell=1}^{n}(\ell+1) = \frac{1}{2}(n^2+3n) \in \mathcal{O}(n^2)$ queries. □

VI. Recovery with Non-Adaptive DTW Distance Oracle Queries

A. Hardness Result without Extra Characters

Theorem VI.1 (Indistinguishable Sequences by Binary Queries with DTW Oracle)**.** *There exists a pair of input sequences s and s' such that for any query sequence q, $d_{\mathrm{DTW}}(s,q) = d_{\mathrm{DTW}}(s',q)$. That is, s and s' cannot be distinguished by DTW Distance Oracle queries without extra characters.*

Proof. We can prove this theorem by constructing a witness pair of input sequences. Consider the following pair of input sequences: $s = 010110$ and $s' = 011010$. We argue that this pair of input sequences cannot be distinguished by any binary sequence query q.

First, for query sequences that only consist of 0, it is obvious $d_{\mathrm{DTW}}(s,q) = d_{\mathrm{DTW}}(s',q) = 3$. Then we only need to consider query sequences containing 1('s). To see $d_{\mathrm{DTW}}(s,q) =$

Algorithm 1: Exact Recovery Algorithm via Queries to an Edit Distance Oracle

```
Input: Non-adaptive query sequences $\mathcal{Q} = \{q^{(1)}, q^{(2)}, \ldots, q^{(n+1)}\}$; The edit distance query result from the sequence for
       recovery to each query sequence $\mathcal{R} = \{d^{(1)}, d^{(2)}, \ldots, d^{(n+1)}\}$.
Output: The sequence for recovery $s$.
Function RecoveryEdit($\mathcal{Q}, \mathcal{R}$):
    sequence = []                                                    ▷ Initialize the sequence for recovery.
    sequence.append($n - d^{(2)}$)
    for $i \in [2, n+1]$ do
        sequence.append($d^{(i)} - d^{(i+1)}$)
    sequence = $\phi$ if $d^{(1)} = 0$, else sequence[1, $d^{(1)}$]   ▷ $d^{(1)}$ is the distance to the empty string.
    return s := sequence
```

$d_{\mathrm{DTW}}(s', q)$ in this case, we will show (1) $d_{\mathrm{DTW}}(s, q) \leq d_{\mathrm{DTW}}(s', q)$ and (2) $d_{\mathrm{DTW}}(s, q) \geq d_{\mathrm{DTW}}(s', q)$ hold simultaneously.

Note that the sequence s contains three 1's. To prove case (1), we show that there exists an optimal DTW matching satisfying the following properties:

a) The first 1 in s is matched to a *substring (c.f. Definition II.3)* of q that begins and ends with both 1's;
b) The second 1 in s is matched to a substring of q that begins with 1;
c) The third 1 in s is matched to a substring of q that ends with 1.

To see the existence of such an optimal matching, we would like to show that, if any one of these properties is violated, we can find another matching with at most the same cost that does not violate these properties. We take property a) as an example to illustrate this. If a) is violated, then the substring in q that the first 1 in s gets matched to contains at least a 0 in the beginning or the end, or both. If this substring contains both a 0 and a 1, then we can map the 0 at the beginning (or in the end) to the 0 on the left (or right) side to the first 1 to obtain a matching with lower cost. We consider the substring that contains only 0. In the optimal matching, the first 1 in s cannot get matched to more than one 0 because this will yield more cost than necessary. Then it reduces to the case where 1 is matched to a single 0. In this case, if the left 0 in s is matched to a substring that contains at least a 1 in q, then matching the first 1 (in s) to this (these) 1('s) leads to a matching with lower cost, since the right 0('s) in the substring can be matched to the 0 on the right to the first 1 in s. Then this leaves the discussion for the case that the first "01" in s is matched to a substring with only 0('s) in q. For ease of presentation, we denote this substring by "ss-0". The second 0 in s can always be matched to ss-0 because this will not yield cost and therefore we know in a potential optimal matching the first "010" can be matched to ss-0. Since we know q contains at least a single 1, this(these) 1('s) will be matched to character after the first "010" (i.e., the second 1) in s. We then argue that we can change this matching to obtain an equally optimal matching without violating the properties: i) the first 0 in s is matched to the substring before the first 1 in q; ii) the first 1 and the second 0 in s are simultaneously matched to the first 1 in q; iii) after the first "010" in s, the matching does not change. In this new matching, there is a cost of 1 saved and generated due to the matching changing on the first 1 and the second 0 in s, and therefore the overall DTW cost does not change and the matching remains optimal. For the rest of the properties, the cases and proofs are similar. We therefore omit the detailed analyses.

Next, we will show that, given the matching (between s and q) with these three properties, we can find a matching between s' and q that will generate DTW cost at most c (that is, $d_{\mathrm{DTW}}(s', q) \leq c$). In particular, we give the following reduction in two matchings.

a) All 0's in s' get matched to the same substring in q as all 0's in s;
b) The first 1 and the second 1 in s' get matched to the substring in q that matches the first 1 in s;
c) The third 1 in s' gets matched to the two substrings in q that match the second 1 and the third 1 in s.

By this matching, the cost between s' and q is exactly c. We do not need to know if this matching is optimal for $d_{\mathrm{DTW}}(s', q)$ but this shows $d_{\mathrm{DTW}}(s', q) \leq c$. We note that these three properties hold for any query sequence that contains at least a single 1, because the single 1 can be a substring of this query sequence to satisfy the properties. Thus, this analysis covers all possible cases of a binary query sequence. By symmetry, a similar construction can be shown for the opposite side and the conclusion is $d_{\mathrm{DTW}}(s, q) \geq d_{\mathrm{DTW}}(s', q)$. Combining the two parts of the proof, we obtain that $d_{\mathrm{DTW}}(s, q) = d_{\mathrm{DTW}}(s', q)$ for any binary query q. □

B. Recovery without Extra Characters w.r.t. Equivalence Classes

As indicated by Theorem VI.1, there exist input sequences that cannot be distinguished by DTW distance oracle queries. For ease of presentation, we say that any two different input sequences s and s' are *distinguishable* if s and s' can be distinguished by DTW Distance Oracle queries. We categorize mutually indistinguishable sequences into *equivalence classes*. In this context, using binary queries, the best solution we can provide in this problem setting is to recover those input sequences up to their equivalence class.

The characterization of the set of *indistinguishable* binary sequences, given a parameterized sequence length n, is not so simple to describe (which can be seen from Observation 4 of [26]). However, we can propose an optimal query strategy in this setting to distinguish all distinguishable sequences and *prove optimality* by making use of the reduction between the calculation of DTW distance and the min 1-separated sum problem [25], [26]. We introduce the necessary results from [26] below and interpret them in our setting.

Definition VI.1 (Min 1-Separated Sum (MSS), [26])**.** The min 1-separated sum (MSS) problem takes the inputs of a sequence $(b_1, \dots, b_m)$ of m positive integers and an integer $r \geq 0$. The problem is to select r integers $b_{i_1}, \dots, b_{i_r}$ with $1 \leq i_1 < i_2 < \dots < i_r \leq m$ and $i_j < i_{j+1} - 1$ for all $1 \leq j < r$ such that $\sum_{j=1}^{r} b_{i_j}$ is minimized. We say $((b_{i_1}, \dots, b_{i_r}), r)$ is an MSS instance.

Theorem VI.2 (DTW-to-MSS Reduction, [26], Theorem 2)**.** *Let $x \in \{0,1\}^m$ and $y \in \{0,1\}^n$ be two binary strings such that $x[1] = y[1], x[m] = y[n]$, and $\#\textsc{runs}(x) \geq \#\textsc{runs}(y)$. Then, the DTW distance between x and y, i.e., $d_{\mathrm{DTW}}(x,y)$, equals the sum of a solution for* $\mathrm{MSS}\Big(\big(\textsc{lor}(x,2), \dots, \textsc{lor}(x, \#\textsc{runs}(x) - 1)\big), (\#\textsc{runs}(x) - \#\textsc{runs}(y))/2\Big)$.

For ease of presentation, we will use $\mathrm{MSS}(x, (\#\textsc{runs}(x) - \#\textsc{runs}(y))/2)$ to represent the same MSS instance.

Theorem VI.3 ([26], Observation 4)**.** *Let $x \in \{0,1\}^m, y \in \{0,1\}^n$ with $m' := \#\textsc{runs}(x) \geq n' := \#\textsc{runs}(y)$. Further, let $a := \textsc{lor}(x,1), a' := \textsc{lor}(x,m'), b := \textsc{lor}(y,1)$, and $b' := \textsc{lor}(y,n')$. The following holds:*
If $x[1] \neq y[1]$, then:

$$
\begin{aligned}
&d_{\mathrm{DTW}}(x,y) \\
&= \begin{cases} \max(a,b), & m' = n' = 1; \\ a + d_{\mathrm{DTW}}(x[a+1,m], y), & m' > n' = 1; \\ \min\big(a + d_{\mathrm{DTW}}(x[a+1,m], y), \\ \qquad b + d_{\mathrm{DTW}}(x, y[b+1,n])\big) & n' > 1. \end{cases}
\end{aligned}
$$

If $x[1] = y[1]$ and $x[m] \neq y[n]$, then:

$$
\begin{aligned}
&d_{\mathrm{DTW}}(x,y) \\
&= \begin{cases} a' + d_{\mathrm{DTW}}\left(x\left[1, m-a'\right], y\right), & n' = 1; \\ \min\big(a' + d_{\mathrm{DTW}}\left(x\left[1, m-a'\right], y\right), \\ \qquad b' + d_{\mathrm{DTW}}\left(x, y\left[1, n-b'\right]\right)\big) & n' > 1. \end{cases}
\end{aligned}
$$

In Theorem VI.3, we call a, b, a' and b' (which are the length of first/last blocks of x or y) *offsets*. Theorem VI.3 actually states that, for two sequences x, y with different starting and ending characters, by removing the first/last run of x or y, calculating $d_{\mathrm{DTW}}(x,y)$ can be reduced to calculating the offset and solving a DTW sub-problem where the sub-sequences start and end with the same character.

To illustrate how we can transfer a DTW problem to an MSS instance, we give a concrete example here. Let $s = 010110$, $q^{(1)} = 010$, $q^{(2)} = 011$. We first consider the calculation of the DTW distance between s and $q^{(1)}$. Since the first and the last blocks of s and $q^{(1)}$ are the same and the number of runs of s is more than that of $q^{(1)}$, we can directly apply Theorem VI.2, where we have the MSS instance $\mathrm{MSS}((1,1,2),1)$ and the DTW distance is equal to the solution to this MSS instance. As for the computation of the DTW distance between s and $q^{(2)}$, we need to first apply Theorem VI.3 since the last blocks of s and $q^{(2)}$ are different. By Theorem VI.3, $d_{\mathrm{DTW}}(s, q^{(2)}) = \min(1 + d_{\mathrm{DTW}}(\text{“01011”}, q^{(2)}), 2 + d_{\mathrm{DTW}}(s, \text{“0”}))$. Then by Theorem VI.2, the calculation of $d_{\mathrm{DTW}}(\text{“01011”}, q^{(2)})$ yields the MSS instance $\mathrm{MSS}((1,1),1)$ and computing $d_{\mathrm{DTW}}(s, \text{“0”}))$ is equivalent to $\mathrm{MSS}((1,1,2),2)$.
We now show the lower bound on the query complexity using binary queries.

Theorem VI.4 (Lower Bound for DTW Equivalence Class Recovery)**.** *For binary alphabet $\{0,1\}$, any algorithm to recover an arbitrary input sequence $s \in \{0,1\}^\ell$, where $0 \leq \ell \leq n$, up to equivalence class, by querying the DTW distance to a set of sequences, requires a query complexity of $\Omega(n)$.*

Proof. We will assume the input sequence is of length $\leq n$ and all the query sequences are of length $\mathcal{O}(n)$, when the context is clear in the proof.

Claim VI.5. Given $\mathrm{MSS}_1((\underbrace{3,\dots,3}_{\mathrm{a}},1,3,3,\underbrace{3,\dots,3}_{\mathrm{b}}),x)$ and $\mathrm{MSS}_2((\underbrace{3,\dots,3}_{\mathrm{a}},2,3,2,\underbrace{3,\dots,3}_{\mathrm{b}}),x)$, we claim that when $x = 1$, $\mathrm{MSS}_1 \neq \mathrm{MSS}_2$ and when $2 \leq x \leq (a+b+4)/2$, $\mathrm{MSS}_1 = \mathrm{MSS}_2$.

Proof of claim VI.5. By the definition of MSS, when $x = 1$, $\mathrm{MSS}_1 = \min(3, \dots, 3, 1, 3, 3, 3, \dots, 3) = 1$ and $\mathrm{MSS}_2 = \min(3, \dots, 3, 2, 3, 2, 3, \dots, 3) = 2$. When $2 \leq x < (a+b+4)/2$, $\mathrm{MSS}_1 = 3x - 2 = \mathrm{MSS}_2$. When $x = (a+b+4)/2$, if a is odd, $\mathrm{MSS}_1 = 3x = \mathrm{MSS}_2$; otherwise, $\mathrm{MSS}_1 = 3x - 2 = \mathrm{MSS}_2$.

Claim VI.6. Let $\mathcal{Q}$ be a query set which can distinguish any pair of binary input sequences that are distinguishable. For $\forall c \in \mathbb{N}_+$ such that $6c + 9 \leq n$, $\exists q \in \mathcal{Q}$ such that $\#\textsc{runs}(q) \in [2c, 2c+6]$.

Proof of claim VI.6. Consider two input sequences, $s = 01^3 01^3 (0^3 1^3)^c 0$ and $s' = 01^3 0^2 1^3 0^2 1^3 (0^3 1^3)^{c-1} 0$, where $\mathrm{len}(s) = \mathrm{len}(s') = 6c + 9 \leq n$. We know that $\#\textsc{runs}(s) = \#\textsc{runs}(s') = 2c+5$. First we show that s and s' are distinguishable. Let $q^\dagger = 0(10)^c 10$, where $\#\textsc{runs}(q^\dagger) = 2c+3$. According to Theorem VI.2, $d_{\mathrm{DTW}}(s, q^\dagger) = \mathrm{MSS}((3,1,3,3,\dots),1) = 1$ and $d_{\mathrm{DTW}}(s', q^\dagger) = \mathrm{MSS}((3,2,3,2,3,\dots),1) = 2$. Thus, $q^\dagger$ can distinguish s and s'.

Next we show, for any query q such that $\#\textsc{runs}(q) \geq 2c + 7$ or $\leq 2c - 1$, $d_{\mathrm{DTW}}(s,q) = d_{\mathrm{DTW}}(s',q)$. Note that, to compute the DTW distances, according to Theorem VI.3, we may remove the first/last blocks of s (and s') or q to reduce to the case of Theorem VI.2. Since s and s' have the same first and last blocks, the offsets while reducing to the case of Theorem VI.2 are the same. To prove that $d_{\mathrm{DTW}}(s,q) = d_{\mathrm{DTW}}(s',q)$, we only need to prove that for each possible reduction, the corresponding reduced MSS instances have the same sum of solutions (see Example VI.1 for illustration).

Example VI.1. *To illustrate, take $c = 2$ and $q = 101$. In this case, we would have $s = 01^3 01^3 0^3 1^3 0^3 1^3 0$, $s' = 01^3 0^2 1^3 0^2 1^3 0^3 1^3 0$. According to Theorem VI.3, we have $d_{\text{DTW}}(s, q) = \min(1 + d_{\text{DTW}}(s[2, \text{len}(s)], q), 1 + d_{\text{DTW}}(s, q[2, \text{len}(q)])) = \min(1 + d_{\text{DTW}}(s[2, 21], q), 1 + d_{\text{DTW}}(s, q[2, 3]))$. Then $d_{\text{DTW}}(s[2, 21], q) = \min(1 + d_{\text{DTW}}(s[2, 20], q), 1 + d_{\text{DTW}}(s[2, 21], q[1, 2]))$. Also, we note that $d_{\text{DTW}}(s, q[2, 3])) = \min(1 + d_{\text{DTW}}(s[1, 20], q[2, 3]), 1 + d_{\text{DTW}}(s, q[2]))$.*

Therefore, to show that $d_{\text{DTW}}(s, q) = d_{\text{DTW}}(s', q)$, we only need to prove that

$$\begin{cases} d_{\text{DTW}}(s[2, 21], q) = d_{\text{DTW}}(s'[2, 21], q); \\ d_{\text{DTW}}(s[2, 20], q[1, 2]) = d_{\text{DTW}}(s'[2, 21], q[1, 2]); \\ d_{\text{DTW}}(s[1, 20], q[2, 3]) = d_{\text{DTW}}(s'[1, 20], q[2, 3]); \\ d_{\text{DTW}}(s, q[2, 3]) = d_{\text{DTW}}(s', q[2, 3]), \end{cases}$$

where each of the 4 cases corresponds to an MSS instance.

Suppose after applying Theorem VI.3, s and q are reduced to sub-sequences s^* and q^* (where s^* and q^* have the same beginning and ending characters), while s' and q are reduced to s'^* and q^*. Now we calculate $d_{\text{DTW}}(s^*, q^*)$ and $d_{\text{DTW}}(s'^*, q^*)$ according to Theorem VI.2. Suppose s^* and s'^* have k^* runs and q^* have l^* runs.

Case 1. If $\#\text{RUNS}(q) \geq 2c + 7 = \#\text{RUNS}(s) + 2$, then $k^* \leq l^*$, by Theorem VI.2 the generated MSS instance only depends on q^* and k^*. Thus, $d_{\text{DTW}}(s^*, q^*) = d_{\text{DTW}}(s'^*, q^*)$.

Case 2. If $\#\text{RUNS}(q) \leq 2c - 1$, then $k^* > l^*$ and we have the MSS instances $\text{MSS}(s^*, (k^* - l^*)/2)$ and $\text{MSS}(s'^*, (k^* - l^*)/2)$. Note that, $(k^* - l^*)/2 \geq (\#\text{RUNS}(s) - \#\text{RUNS}(q) - 2)/2 \geq ((2c + 5) - (2c - 1) - 2)/2 = 2$. By Claim VI.5, we have $\text{MSS}(s^*, (k^* - l^*)/2) = \text{MSS}(s'^*, (k^* - l^*)/2)$. Thus, $d_{\text{DTW}}(s^*, q^*) = d_{\text{DTW}}(s'^*, q^*)$.

Combining case 1 and case 2, we know $d_{\text{DTW}}(s, q) = d_{\text{DTW}}(s', q)$ when $\#\text{RUNS}(q) \geq 2c + 7$ or $\leq 2c - 1$. Since there always exists $q \in \mathcal{Q}$ that can distinguish s and s', we know that $\#\text{RUNS}(q) \in [2c, 2c + 6]$, which proves the claim.

Let $c' \in \mathbb{N}_+$ satisfy $24c' - 9 \leq n$. Let $c = 4c' - 3$. We have $6c + 9 \leq n$. By Claim VI.6, $\exists q \in \mathcal{Q}$ such that $\#\text{RUNS}(q) \in [2c, 2c+6]$, i.e., $\#\text{RUNS}(q) \in [8c' - 6, 8c']$. For $c' = 1, 2, \ldots$, intervals $[8c' - 6, 8c']$ are disjoint. Therefore, there should be at least $\lfloor (n + 9)/24 \rfloor = \Omega(n)$ queries in the set $\mathcal{Q}$. □

With these useful results at hand, now we prove the following results for recovering sequences using the DTW distance oracle with only binary queries.

Theorem VI.7 (Non-adaptive Strategy for DTW Equivalence Class Recovery)**.** *There exists a set $\mathcal{Q}$ of $2n \in \mathcal{O}(n)$ queries, each of which has $\mathcal{O}(n)$ length, such that for any two different input sequences s and s', s and s' are distinguishable $\iff$ s and s' can be distinguished by $\mathcal{Q}$.*

Proof. First ($\Leftarrow$), for any given query set $\mathcal{Q}$ and two different input sequences s and s', if s and s' can be distinguished by $\mathcal{Q}$ then s and s' are distinguishable. Then we need to prove the opposite side ($\Rightarrow$). To see this, we construct the following query set $\mathcal{Q}$ of size $2n \in \mathcal{O}(n)$ and prove the contrapositive: if s and s' cannot be distinguished by $\mathcal{Q}$, then s and s' are not distinguishable.

Let

$$z_i = \begin{cases} 0^n, \ i = 1; \\ 0^n 1 (01)^{m-1} 0^n, \ i = 2m + 1; \\ 0^n (10)^{m-1} 1^n, \ i = 2m, \end{cases}$$

and

$$o_i = \begin{cases} 1^n, \ i = 1; \\ 1^n 0 (10)^{m-1} 1^n, \ i = 2m + 1; \\ 1^n (01)^{m-1} 0^n, \ i = 2m, \end{cases}$$

where $1 \leq i \leq n$ and m is an positive integer. It is clear that o_i's and z_i's are of $\mathcal{O}(n)$ length. Let $\mathcal{Q} = \{o_i | 1 \leq i \leq n\} \bigcup \{z_i | 1 \leq i \leq n\}$. We show that given any two different input sequences s and s', if s and s' cannot be distinguished by $\mathcal{Q}$ then s and s' are not distinguishable.

Claim VI.8. Given two different input sequences s and s', if the condensed expressions of s and s' are different, then s and s' can be distinguished by $\mathcal{Q}$.

Proof of claim VI.8. We note that the condensed expressions of o_i and z_i for $1 \leq i \leq n$ cover all possible condensed expressions for a sequence with length at most n. Therefore, for input sequence s, we can find a query sequence $q \in \mathcal{Q}$ such that s and q have the same condensed expression, and we would have $d_{\text{DTW}}(s, q) = 0$. Since s and s' have different condensed expressions, we would have $d_{\text{DTW}}(s', q) \neq 0$. Thus, q distinguishes s and s'.

Suppose s and s' cannot be distinguished by $\mathcal{Q}$. By Claim VI.8, we know that the condensed expression of s and s' are the same. Let the number of runs in s and s' be $k = \#\text{RUNS}(s) = \#\text{RUNS}(s')$.

Claim VI.9. If s and s' cannot be distinguished by $\mathcal{Q}$, then $k \geq 3$.

Proof of claim VI.9. Consider the query sequence $z_1 = 0^n$ and $o_1 = 1^n$. By querying z_1 and o_1, we can obtain the number of 1's and 0's in the input sequence. If $k \leq 2$, then s (and s') would contain at most a single 0-run and a 1-run. With queries z_1 and o_1 we can determine the length of the 0-run and the 1-run in s and s', and therefore distinguish them.

Claim VI.10. If s and s' cannot be distinguished by $\mathcal{Q}$, then $\text{LOR}(s, 1) = \text{LOR}(s', 1)$ and $\text{LOR}(s, k) = \text{LOR}(s', k)$.

Proof of claim VI.10. If s and s' start with 0, we show that $d_{\text{DTW}}(s, o_{k-1}) = \text{LOR}(s, 1)$. Since $k \geq 3$, by Theorem VI.3,

$$\begin{aligned} &d_{\text{DTW}}(s, o_{k-1}) = \\ &\quad \min\left(d_{\text{DTW}}(s[\text{LOR}(s, 1) + 1, \text{len}(s)], o_{k-1}) + \text{LOR}(s, 1),\right. \\ &\qquad \left. n + d_{\text{DTW}}(s, o_{k-1}[n + 1, \text{len}(o_{k-1})])\right) \end{aligned}$$

Note that $d_{\text{DTW}}(s[\text{LOR}(s, 1) + 1, \text{len}(s)], o_{k-1}) = 0$, and $\text{LOR}(s, 1) \leq n \leq n + d_{\text{DTW}}(s, o_{k-1}[n + 1, \text{len}(o_{k-1})])$, we know that $d_{\mathrm{DTW}}(s, o_{k-1}) = \textsc{lor}(s, 1)$. Similarly $d_{\mathrm{DTW}}(s', o_{k-1}) = \textsc{lor}(s', 1)$. Since s and s' cannot be distinguished by $\mathcal{Q}$, we have $\textsc{lor}(s, 1) = d_{\mathrm{DTW}}(s, o_{k-1}) = d_{\mathrm{DTW}}(s', o_{k-1}) = \textsc{lor}(s', 1)$. Similarly, we would have $\textsc{lor}(s, k) = d_{\mathrm{DTW}}(s, z_{k-1}) = d_{\mathrm{DTW}}(s', z_{k-1}) = \textsc{lor}(s', k)$.

By symmetry, if s and s' starts with 1, we would have $\textsc{lor}(s, 1) = d_{\mathrm{DTW}}(s, z_{k-1}) = d_{\mathrm{DTW}}(s', z_{k-1}) = \textsc{lor}(s', 1)$ and $\textsc{lor}(s, k) = d_{\mathrm{DTW}}(s, o_{k-1}) = d_{\mathrm{DTW}}(s', o_{k-1}) = \textsc{lor}(s', k)$. This finishes the proof for Claim VI.10.

Next, we show that s and s' cannot be distinguished by any binary query r. Let the number of runs in r be $l = \#\textsc{runs}(r)$. Given s and r, we can calculate $d_{\mathrm{DTW}}(s, r)$ with Theorem VI.3 and Theorem VI.2. Note that in Theorem VI.3, we may remove the first/last blocks of s (and s') or r to reduce to the case of Theorem VI.2. By Claim VI.10 we have $\textsc{lor}(s, 1) = \textsc{lor}(s', 1)$ and $\textsc{lor}(s, k) = \textsc{lor}(s', k)$, while $\textsc{lor}(r, 1)$ and $\textsc{lor}(r, l)$ are only related to r but not s and s'. Therefore, the offsets while reducing to the case of Theorem VI.2 are the same. To prove that $d_{\mathrm{DTW}}(s, r) = d_{\mathrm{DTW}}(s', r)$, we only need to prove that for each possible reduction, the corresponding reduced MSS instances have the same sum of solutions (see Example VI.2 for illustration).

Example VI.2. *To illustrate, take $s = 010110$, $s' = 011010$ and $r = 1001011$ as an example. In this case, we would have $\textsc{lor}(s, 1) = \textsc{lor}(s', 1) = 1$, $\textsc{lor}(s, k) = \textsc{lor}(s', k) = 1$, $\textsc{lor}(r, 1) = 1$ and $\textsc{lor}(r, l) = 2$. According to Theorem VI.3, we have $d_{\mathrm{DTW}}(s, r) = \min(\textsc{lor}(s, 1) + d_{\mathrm{DTW}}(s[\textsc{lor}(s, 1) + 1, \mathrm{len}(s)], r), \textsc{lor}(r, 1) + d_{\mathrm{DTW}}(s, r[\textsc{lor}(r, 1) + 1, \mathrm{len}(r)])) = \min(1 + d_{\mathrm{DTW}}(s[2, 6], r), 1 + d_{\mathrm{DTW}}(s, r[2, 7]))$. Then $d_{\mathrm{DTW}}(s[2, 6], r) = \min(\textsc{lor}(s, k) + d_{\mathrm{DTW}}(s[2, 6 - \textsc{lor}(s, k)], r), \textsc{lor}(r, l) + d_{\mathrm{DTW}}(s[2, 6], r[1, \mathrm{len}(r) - \textsc{lor}(r, l)])) = \min(1 + d_{\mathrm{DTW}}(s[2, 5], r), 2 + d_{\mathrm{DTW}}(s[2, 6], r[1, 5]))$. Also, $d_{\mathrm{DTW}}(s, r[2, 7])) = \min(\textsc{lor}(s, k) + d_{\mathrm{DTW}}(s[1, \mathrm{len}(s) - \textsc{lor}(s, k)], r[2, 7]), \textsc{lor}(r, k) + d_{\mathrm{DTW}}(s, r[2, 7 - \textsc{lor}(r, k)])) = \min(1 + d_{\mathrm{DTW}}(s[1, 5], r[2, 7]), 2 + d_{\mathrm{DTW}}(s, r[2, 5]))$.*

Therefore, to show that $d_{\mathrm{DTW}}(s, r) = d_{\mathrm{DTW}}(s', r)$, we only need to prove that

$$\begin{cases} d_{\mathrm{DTW}}(s[2,5], r) = d_{\mathrm{DTW}}(s'[2,5], r); \\ d_{\mathrm{DTW}}(s[2,6], r[1,5]) = d_{\mathrm{DTW}}(s'[2,6], r[1,5]); \\ d_{\mathrm{DTW}}(s[1,5], r[2,7]) = d_{\mathrm{DTW}}(s'[1,5], r[2,7]); \\ d_{\mathrm{DTW}}(s, r[2,5]) = d_{\mathrm{DTW}}(s', r[2,5]), \end{cases}$$

where each of the 4 cases corresponds to an MSS instance. □

Suppose after applying Theorem VI.3, s and r are reduced to sub-sequences s^* and r^* (where s^* and r^* have the same beginning and ending characters), while s' and r are reduced to s'^* and r^*. Now we calculate $d_{\mathrm{DTW}}(s^*, r^*)$ and $d_{\mathrm{DTW}}(s'^*, r^*)$ according to Theorem VI.2. Suppose s^* and s'^* have k^* runs and r^* have l^* runs.

Case 1. If $k^* = l^*$, then $d_{\mathrm{DTW}}(s^*, r^*) = d_{\mathrm{DTW}}(s'^*, r^*) = 0$.

Case 2. If $k^* < l^*$, by Theorem VI.2 the generated MSS instance only depends on r^* and k^*. Thus, $d_{\mathrm{DTW}}(s^*, r^*) = d_{\mathrm{DTW}}(s'^*, r^*)$.

Case 3. If $k^* > l^*$, we have the MSS instances $\mathrm{MSS}(s^*, (k^* - l^*)/2)$ and $\mathrm{MSS}(s'^*, (k^* - l^*)/2)$. Note that, we can always find a query $q \in \mathcal{Q}$ which has l^* runs and has the same starting and ending characters as s^* and s'^*. Consider $d_{\mathrm{DTW}}(s, q)$ and $d_{\mathrm{DTW}}(s', q)$. Note that the first and last runs of q are both of length n and removing them would yield at least cost n, the only possible reduction would be $d_{\mathrm{DTW}}(s^*, q)$ and $d_{\mathrm{DTW}}(s'^*, q)$. Since q cannot distinguish s and s', we have $d_{\mathrm{DTW}}(s, q) = d_{\mathrm{DTW}}(s', q)$, so $d_{\mathrm{DTW}}(s^*, q) = d_{\mathrm{DTW}}(s'^*, q)$, implying that $\mathrm{MSS}(q^*, (k^* - l^*)/2)$ and $\mathrm{MSS}(q'^*, (k^* - l^*)/2)$ have the same sum of solution. Therefore, $d_{\mathrm{DTW}}(s^*, r^*) = d_{\mathrm{DTW}}(s'^*, r^*)$.

Combining the 3 cases above, we always have $d_{\mathrm{DTW}}(s^*, r^*) = d_{\mathrm{DTW}}(s'^*, r^*)$, so $d_{\mathrm{DTW}}(s, r) = d_{\mathrm{DTW}}(s', r)$, implying that s and s' cannot be distinguished by r. This finishes the proof for Theorem VI.7. □

C. Exact Recovery with Extra Character(s)

Theorem VI.11 (Lower Bound for DTW Exact Recovery)**.** *For a binary alphabet $\{0, 1\}$, any algorithm to recover arbitrary input sequence $s \in \{0, 1\}^\ell$ where $0 \leq \ell \leq n$ by querying DTW distance to a set of sequences of length $\mathcal{O}(n)$ from a constant-sized extended alphabet Σ would require a query complexity of $\Omega(n/\log n)$.*

Theorem VI.11 shows the lower bound of the query complexity for DTW exact recovery. The proof of Theorem VI.11 is given by an information-theoretic lower bound, which refers back to the proof of Theorem V.1.

With this lower bound, now we would like to show that if one is allowed to construct queries from a slightly larger alphabet beyond $\{0, 1\}$, there exists a non-adaptive query strategy such that this lower bound is attainable as per the order of magnitude.

1) With One Extra Character:

Theorem VI.12 (Non-adaptive Strategy for DTW Exact Recovery with 1 Extra Character)**.** *For a binary alphabet $\{0, 1\}$ and an input sequence $s := \{0, 1\}^\ell$ where $0 \leq \ell \leq n$, there exists an algorithm to recover the input sequence s, given $n^2 + n \in \mathcal{O}(n^2)$ query sequences $\mathcal{Q}$ and the $d_{\mathrm{DTW}}(s, q)$ to each query sequence $q \in \mathcal{Q}$, where the query sequences are allowed to use only one extra character.*

Proof of Theorem VI.12 We give our proof by constructing $n^2 + n$ query sequences of length $\leq n$ and presenting an algorithm to recover an input sequence s from its DTW distance to these $n^2 + n$ query sequences.

Let $z_{i,k} = \begin{cases} 0(10)^m(\frac{1}{2})^k, i = 2m+1; \\ (01)^m(\frac{1}{2})^k, i = 2m, \end{cases}$ and $o_{i,k} = \begin{cases} 1(01)^m(\frac{1}{2})^k, i = 2m+1; \\ (10)^m(\frac{1}{2})^k, i = 2m, \end{cases}$ where m is a non-negative integer, $1 \leq i \leq n$ and $0 \leq k \leq n - i$. Let $\mathcal{Q} = \{o_{i,k} | 1 \leq$

$i \leq n, 0 \leq k \leq n-i\} \bigcup \{z_{i,k} | 1 \leq i \leq n, 0 \leq k \leq n-i\}$. We have $|\mathcal{Q}| = 2\sum_{i=1}^{n} i = n^2 + n$.

Without loss of generality, we can assume that s starts with a 0 and has t runs, where the i-th run of s is $\text{LOR}(s,i)$, $l = \sum_{i=1}^{t} \text{LOR}(s,i)$. Then $d_{\text{DTW}}(z_{t,0}, s) = 0$. Consider $d_{\text{DTW}}(z_{i,k}, s)$ where $1 \leq i \leq t$. The first i runs of s have a total length of $\sum_{j=1}^{i} \text{LOR}(s,j)$, and the last $t-i$ runs of s have a total length of $\sum_{j=i+1}^{t} \text{LOR}(s,j)$.

Claim VI.13. For $1 \leq i \leq t$, $d_{\text{DTW}}(z_{i,k}, s) = \frac{k}{2} \iff k \geq \sum_{j=i+1}^{t} \text{LOR}(s,j)$.

Proof of Claim VI.13. Since each $\frac{1}{2}$ in $z_{i,k}$ corresponds to at least $\frac{1}{2}$ cost, we have $d_{\text{DTW}}(z_{i,k}, s) \geq \frac{k}{2}$. Note that $s[1, \sum_{j=1}^{i} \text{LOR}(s,j)]$ and $z_{i,k}[1,i]$ can be perfectly matched. If $k \geq \sum_{j=i+1}^{t} \text{LOR}(s,j)$, then $s[(\sum_{j=1}^{i} \text{LOR}(s,j)) + 1, l]$ and $z_{i,k}[i+1, i+k] = (\frac{1}{2})^k$ can be matched with exactly $\frac{k}{2}$ cost, so $d_{\text{DTW}}(z_{i,k}, s) = \frac{k}{2}$. Otherwise, if $k < \sum_{j=i+1}^{t} \text{LOR}(s,j)$, we show that $d_{\text{DTW}}(z_{i,k}, s) > \frac{k}{2}$. In fact, if any of the $\frac{1}{2}$ in $z_{i,k}$ is matched to more than one character in s, we would already have $d_{\text{DTW}}(z_{i,k}, s) > \frac{k}{2}$. If all $\frac{1}{2}$'s in $z_{i,k}$ have degree 1, then $z_{i,k}[1..i]$ must be matched with $s[1, l-k]$. Since $l - k > \sum_{j=1}^{i} \text{LOR}(s,j)$, $z_{i,k}[1..i]$ and $s[1, l-k]$ cannot be perfectly matched, yielding a non-zero cost. This finishes the proof of Claim VI.13.

By Claim VI.13, we know that for $1 \leq i \leq t$, $\sum_{j=i+1}^{t} \text{LOR}(s,j) = min_{d_{\text{DTW}}(z_{i,k},s) = \frac{k}{2}} k$. In this way, we can recover the length of each run in s, and therefore recover s. A similar analysis can be performed for the cases where s starts with a single 1. ■

2) With Two Extra Characters:

Theorem VI.14 (Non-adaptive Strategy for DTW Exact Recovery with 2 Extra Characters). *For a binary alphabet $\{0,1\}$ and an input sequence $s := \{0,1\}^{\ell}$ where $0 \leq \ell \leq n$, there exists an algorithm to recover the input sequence s, given $n+2 \in \mathcal{O}(n)$ query sequences $\mathcal{Q}$ of length $\leq n$ and the $d_{\text{DTW}}(s,q)$ to each query sequence $q \in \mathcal{Q}$, where the query sequences are allowed to use only $\mathcal{O}(1)$ extra characters.*

Proof of Theorem VI.14. We give our proof by constructing $n+2 \in \mathcal{O}(n)$ query sequences of length $\leq n$ and presenting an algorithm to recover an input sequence s from its DTW distance to these $n+2$ query sequences.

Note that for any sequence s, we have $d_{\text{DTW}}(s,0) = 0 \iff s$ consists of only 0's and $d_{\text{DTW}}(s,1) = 0 \iff s$ consists of only 1's. We can also derive that $d_{\text{DTW}}(0^m, 1) = m$ and $d_{\text{DTW}}(1^m, 0) = m$. Thus, any input sequence consisting of only 0s or 1s can be exactly recovered by the two query sequences 0 and 1. For simplicity, **we assume in the rest of the proof that the input sequence s contains both 0 and** 1 and let $s = s[1]s[2]\ldots s[\ell]$.

Query Sequences Construction. Let a, b be two fractional characters that satisfy $0 < b - a < a < b < \frac{1}{2}$ and the denominators of a, b are co-prime. We will use a, b as the extra characters to construct the query sequences. In particular, the rest of the query sequences (other than the 0 query and the 1 query) consist of queries $\mathcal{Q}$ is in the form of $a^{n-i}b^{i}$, where $i = 1, \ldots, n$. It is not hard to see this set of queries are monotonic sequences, for which we show the following property holds in the distance query to DTW. We will use $a = \frac{1}{3}$ and $b = \frac{2}{5}$ as a running example for better explanation when necessary, but the proof works for all a, b satisfying the condition.

Lemma VI.15. *Given a monotonic sequence q of length n where*

$$\min_{i \in [n]} \max\{|q[i] - 0|, |q[i] - 1|\} > \max_{i,j \in [n]} |q[i] - q[j]|, \quad (2)$$

for any input sequence s with length $\ell \leq n$, given a DTW matching M for (q,s), we have $\deg(q[i]) = 1$ *for all elements $q[i]$ in q.*

Proof. Suppose $\exists i \in [\ell]$ such that $\deg(q[i]) > 1$. We first prove the following two claims.

Claim VI.16. For any edge e in M, the two vertices corresponding to e in M cannot have degree > 1 at the same time.

Proof of Claim VI.16. This is trivial, since otherwise by deleting e we would obtain a better matching.

Claim VI.17. There does not exist $i \in [n]$ and $j \in [\ell]$ such that $\deg(q[i]) > 1$ and $\deg(s[j]) > 1$.

Proof of Claim VI.17. We prove this by contradiction. Suppose $\exists (i,j)$ where $i \in [n]$ and $j \in [\ell]$ such that $\deg(q[i]) > 1$ and $\deg(s[j]) > 1$. Consider the following index sets $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$. Let $\mathbf{X} = \{x \in [n] \mid \deg(q[x]) > 1\}$, $\mathbf{Y} = \{y \in [\ell] \mid \deg(s[y]) > 1\}$ and $\mathbf{Z} = \{z \in [\ell] \mid \exists x \in \mathbf{X}$ such that edge $(q[x], s[z]) \in M\}$. According to Claim VI.16, we know that $\mathbf{Y} \bigcap \mathbf{Z} = \emptyset$. Let $d = \min_{y \in \mathbf{Y}, z \in \mathbf{Z}} |y - z|$, we would have $d > 0$. Suppose we have $x_0 \in \mathbf{X}, y_0 \in \mathbf{Y}, z_0 \in \mathbf{Z}$ such that edge $(q[x_0], s[z_0]) \in M$ and $|y_0 - z_0| = d$. And this leaves two cases to discuss.

1) if $y_0 < z_0$, then we know that (i) $\forall k \in (y_0, z_0)$, $\deg(s[k]) = 1$; otherwise we would have $k \in \mathbf{Y}$ and $|k - z_0| < d$, causing a contradiction. (ii) $\forall k_1, k_2$ within range (y_0, z_0), $k_1 \neq k_2$, we would have $s[k_1]$ and $s[k_2]$ matched to different vertices in q; otherwise, suppose edges $(q[t], s[k_1]) \in M$ and $(q[t], s[k_2]) \in M$. We would have $t \in \mathbf{X}$, $k_1 \in \mathbf{Z}$ and $|y_0 - k_1| < d$, causing a contradiction.

Now, since d is minimal, we can suppose that $s[y_0]$ is matched to

$$\{q[w - \deg(s[y_0]) + 1], q[w - \deg(s[y_0]) + 2], \ldots, q[w]\},$$

and $q[x_0]$ is matched to

$$\{s[z_0], s[z_0 + 1], \ldots, s[z_0 + \deg(q[x_0]) - 1]\}.$$

Since $y_0 < z_0$, by the monotonic property of the matching, we know that $w < x_0$. With (i) and (ii), we know that for $w < l < x_0$ and $y_0 < r < z_0$, the vertices $q[l]$'s and $s[r]$'s are perfectly matched one-to-one. Fig 4 is an illustration of such an example.

We now claim, by re-matching edges between vertices $q[w], q[w+1], \ldots, q[x_0]$ and $s[y_0], s[y_0+1], \ldots, s[z_0]$, we can

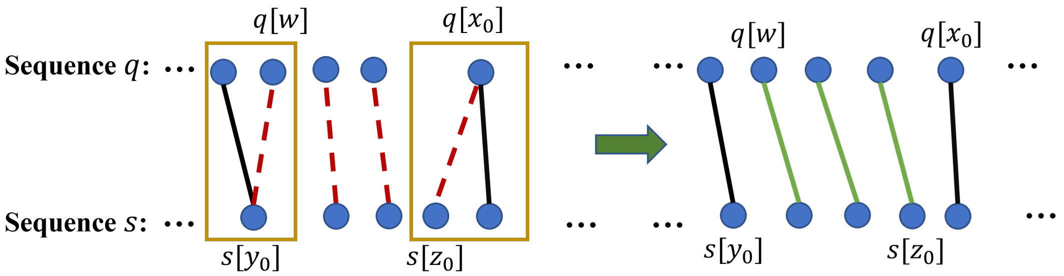

Fig. 4. Optimal re-matching (*hybrid stitching*) when both query and input sequences have a vertex with degree greater than 1 (c.f. Claim VI.17).

construct another matching M' which is better than M, contradicting that M is a DTW matching. We remove the $d+1$ edges $E = \{(q[w], s[y_0]), (q[w+1], s[y_0+1]), \ldots, (q[x_0], s[z_0])\}$ from M and add d new edges $E' = \{(q[w], s[y_0+1]), (q[w+1], s[y_0+2]), \ldots, (q[x_0-1], s[z_0])\}$ to obtain a new matching M'. Since $\deg(s[y_0]) > 1$ and $\deg(q[x_0]) > 1$ in M, M' would still be a valid matching. Computing the sum of two sets of edges E and E', respectively, would yield Equation. 3 (see the cross-column equations).

So M' would be a better matching than M, causing a contradiction.

2) if $y_0 > z_0$, this case is symmetric to 1) and we can use a similar method to complete the proof by contradiction. We give a detailed proof in the appendix.

Combining the two cases finishes the proof for Claim VI.17.

Suppose $\exists i \in [\ell]$ such that $\deg(q[i]) > 1$. With Claim 2, we know that $\forall j \in [\ell]$, $\deg(s[j]) = 1$. Thus, we would have $\sum_{j=1}^{\ell} \deg(s[j]) = \sum_{i=1}^{n} \deg(q[i]) > n \geq \ell = \sum_{j=1}^{\ell} \deg(s[j])$, which causes a contradiction and finishes the proof of Lemma VI.15. □

Furthermore, we have the following lemma for the DTW matching for our query sequences.

Lemma VI.18. *For any given input sequence s and query $q \in \mathcal{Q}$, the DTW matching M for (q, s) has $\deg(s[i]) = 1$ in M if $s[i] = 1$.*

Proof. We give proof by contradiction. Given an optimal DTW matching M for (q, s), suppose $\exists 1 \leq i \leq \ell$ such that $s[i] = 1$ and $\deg(s[i]) > 1$. Suppose $s[i]$ is matched to $q[j], q[j+1], \ldots, q[j + \deg(s[i]) - 1]$.

First, we show that we can "swap" $s[i]$ with its neighboring element while maintaining the optimality of the matching. If one of the neighboring elements of $s[i]$ is 1, w.l.o.g, suppose $s[i+1] = 1$, then we can construct an alternate optimal matching M^* where $\deg(s[i]) = 1$ and $\deg(s[i+1]) > 1$. According to Lemma VI.15, $s[i+1]$ cannot be matched with any of $q[j], q[j+1], \ldots, q[j+\deg(s[i])-1]$ in M, otherwise there would exist $j+1 \leq k \leq j+\deg(s[i])-1$ such that $\deg(q[k]) = 2$. Thus, by matching $q[j+1], \ldots, q[j+\deg(s[i])-1]$ to $s[i+1]$ instead of $s[i]$, we would obtain a new optimal matching M^* where $\deg(s[i]) = 1$ and $\deg(s[i+1]) > 1$.

As there exists at least one 0 in s, we know that there exists an optimal DTW matching M_0^* for (q, s) where $\exists s[i]$ such that $s[i] = 1$, $\deg(s[i]) > 1$ and one of the neighboring element of $s[i]$ is 0. Without loss of generality, suppose $s[i+1] = 0$. Similarly, according to Lemma VI.15, $s[i+1]$ cannot be matched with any of $q[j], q[j+1], \ldots, q[j+\deg(s[i])-1]$ in M_0. Here we construct a new matching M_0' by matching $q[j+1], \ldots, q[j+\deg(s[i])-1]$ to $s[i+1]$ instead of $s[i]$. Fig 5 illustrates an example of such a construction. Considering the total cost of differing edges in both matchings, we have $\sum_{k=j+1}^{j+\deg(s[i])-1} |s[i] - q[k]| > \sum_{k=j+1}^{j+\deg(s[i])-1} \frac{1}{2} > \sum_{k=j+1}^{j+\deg(s[i])-1} |s[i+1] - q[k]|$. Thus M_0^* would be a better matching than M_0, causing a contradiction and thus finishing the proof. □

Notation clarification. For the rest of the proof, we will use $q^{(i)}$ to denote the i-th query in the query set $\mathcal{Q}$ and $q^{(i)}[j]$ the j-th character in $q^{(i)}$.

Lemma VI.19. *For any input sequence s, there exists a set of isomorphic matchings $\mathcal{M}^*$, where $M_i^*(q^{(i)}, s) \in \mathcal{M}^*$ is optimal for query $q^{(i)} \in \mathcal{Q}$.*

Proof. According to previous assumptions, we know that the input sequence s contains at least one 0. Suppose $s[u]$ is the first 0 in s. We construct the following matching M_i^* for each $q^{(i)} \in \mathcal{Q}$:

1) For $1 \leq j < u$, $q^{(i)}[j]$ is matched to $s[j]$ in M_i^*;
2) For $u \leq j \leq u + n - \ell$, $q^{(i)}[j]$ is matched to $s[u]$ in M_i^*;
3) For $u + n - \ell < j \leq n$, $q^{(i)}[j]$ is matched to $s[j - (n - \ell)]$ in M_i^*.

The constructed M_i^*'s form a set of isomorphic matchings, and we will show that each M_i^* is an optimal matching between s and $q^{(i)}$. To prove this, we first define the "shifting" operation.

Definition VI.2 (Shifting Operation for Queries in $\mathcal{Q}$)**.** Given a matching M between input sequence s of length ℓ and query sequence $q \in \mathcal{Q}$ of length n. Suppose $\exists 1 \leq x < y \leq \ell$ s.t. $s[x] = s[y] = 0$, $\deg(s[x]) > 1$, and $\forall x < j < y$, $\deg(s[j]) = 1$. We now construct a new matching M' based on M:

Suppose $q[z]$ is the last character matched to $s[x]$ and $q[w]$ is the first character matched to $s[y]$, we know that $z - x = w - y$ (cf. lemma VI.15). For $x \leq j < y$, we remove the edge $(q[j - x + z], s[j])$ from M and add the edge $(q[j - x + z], s[j+1])$. As $\deg(s[x]) > 1$. This will give us a valid matching. We call this process a *shifting operation*.

An illustration of the shifting operation is shown in Fig 6. The shifting operation reduces $\deg(s[x])$ by 1 and increases $\deg(s[y])$ by 1, while preserving the degree of all other vertices in s. Now we give the following claims for shifting operations.

Claim VI.20. A shifting operation does not reduce the total cost of the matching.

Proof of claim. As one can observe, the shifting operation will not increase the total number of edges – the number of removed edges is equal to the number of newly added edges. Then we only need to consider the cost of those changed edges. Recall that our monotonic query sequences are in the form of $a^{n-k}b^k$ for $k = 1, \dots, n$. To calculate the change of cost in the shifting operation, we have two cases to analyze.
Case 1. All characters between $q[z]$ and $q[w]$ (including $q[z]$ and $q[w]$) in the query sequence are the same, either a or b. In this case, the total cost does not change after the shifting operation. This is because, $\forall s[j] \in s$ s.t. $x < j < y$, the edge changes from $(q[j-x+z], s[j])$ to $(q[j-x+z-1], s[j])$ and the cost $\mathrm{Cost}(q[j-x+z], s[j]) = \mathrm{Cost}(q[j-x+z-1], s[j])$ since $q[j-x+z] = q[j-x+z-1]$. Notice in the matching before shifting, we have the edge $(q[z], s[x])$ while in the matching after shifting this edge is removed but the edge $(q[w], s[y])$ is added. These two edges have equal cost $\mathrm{Cost}(q[z], s[x]) = \mathrm{Cost}(q[w], s[y])$ because $s_x = s_y = 0$.
Case 2. The characters between $q[z]$ and $q[w]$ (including $q[z]$ and $q[w]$) contain both a and b. Without loss of generality, we can assume there exists index i, s.t. for $j < i, q[j] = a$ while for $j \geq i, q[j] = b$. Applying a similar analysis as we did in case 1, the cost of edges containing characters $q[j]$ such that $z < j < i-1$ or $i < j < w$ remains the same after the shifting operation. Suppose $s[t]$ gets matched to $q[i]$ before the shifting operation. We only need to analyze the cost of the (removed and added) edges corresponding to characters $s[x], s[t]$ and $s[y]$. Before the shifting operation, these three characters get matched in edges $(q[z], s[x]), (q[i], s[t])$, respectively, while in the matching after shifting, they are involved in edges $(q[i-1], s[t]), (q[w-1], s[y])$. We can compute the total cost of these three edges before shifting $\mathrm{Cost}_{\text{before}} = |a-0| + |b-s[t]|$ and the total cost after shifting $\mathrm{Cost}_{\text{after}} = |a-s[t]| + |b-0|$. If $s[t] = 0$, then $\mathrm{Cost}_{\text{before}} = a+b = \mathrm{Cost}_{\text{after}}$; otherwise if $s[t] = 1$, then $\mathrm{Cost}_{\text{before}} = a+1-b$ and $\mathrm{Cost}_{\text{after}} = 1-a+b$. Since $0 < a < b < 1$, $\mathrm{Cost}_{\text{before}} < \mathrm{Cost}_{\text{after}}$ when $s[t] = 1$. Therefore in this case, $\mathrm{Cost}_{\text{before}} \leq \mathrm{Cost}_{\text{after}}$.

Combining both cases, the total cost of the matching before the shifting operation could be only less than or equal to the cost after shifting, which proves the claim.

Claim VI.21. Given input sequence s, query $q^{(i)} \in \mathcal{Q}$ and any matching M_i between s and $q^{(i)}$. If M_i satisfies the properties that (i) $\forall 1 \leq j \leq \ell$, $\deg(s[j]) > 1 \Rightarrow s_j = 0$, (ii) $\forall 1 \leq k \leq n$, $\deg(q^{(i)}[k]) = 1$, then we can obtain M_i by applying a series of shifting operations to M_i^*.

Proof of Claim. If the input sequence s contains only a single 0, then this claim is trivial since any matching $M_i = M_i^*$. For cases that the input sequence s contains more than one 0, without loss of generality, we can assume s has k 0's and in the matching M_i, for each 0 in s (denoted by s_{0_m}, $m \in [k]$), the degree $\deg(s_{0_m}) = t_m \geq 1$. Note that, as we defined, the shifting operation can be performed between $s[x]$ and $s[y]$, if $s[x] = s[y] = 0$, $\deg(s[x]) > 1$, and $\forall x < t < y$, $\deg(s[t]) = 1$. This condition obviously holds for the matching M_i^* if $s[x]$ and $s[y]$ are the nearest neighboring 0's in the input sequence s, because all characters between $s[x]$ and $s[y]$ are 1's and in M_i^* all characters $s[j]$ s.t. $s[j] = 1$ we have $\deg(s[j]) = 1$ (indicated by Lemma VI.18). Property (ii) indicates that both M_i^* and M_i have the same number of edges n, and property (i) indicates $\forall 1 \leq j \leq \ell, s[j] = 1 \Rightarrow \deg(s[j]) = 1$. For M_i^* and M_i, we have $\sum_{s[j]=0} \deg(s[j]) = n - (\#1\text{s in } s) = \sum_{m=1}^{k} t_m$. For matching M_i^*, the degree of all 0's is 1 except for the first 0 and therefore the degree of the first 0 is $\sum_{m=1}^{k} t_m - (k-1)$. Therefore, we can perform the shifting operation $\sum_{m=1}^{k} t_m - (k-1) - t_1$ times to move $\sum_{m=1}^{k} t_m - (k-1) - t_1$ edges from the first 0 to the second 0. Similarly, we continue doing shifting operations to move $\sum_{m=j}^{k} t_m - (k-1) - t_j$ edges from the j-th 0 to the $(j+1)$-th 0. We can hence obtain M_i after all shifting operations are finished and this shows the correctness of this claim.

Suppose M_{i0} is an optimal DTW matching between s and $q^{(i)}$. By Lemma VI.15, in DTW matching M_{i0}, $\forall 1 \leq k \leq n, \deg(q^{(i)}[k]) = 1$. By Lemma VI.18 we know that M_{i0} has $\deg(s[i]) = 1$ in M_{i0} if $s[i] = 1$, so $\forall 1 \leq j \leq \ell$, $\deg(s[j]) >$

$$\begin{aligned}
\mathrm{Cost}(E) &= |s[y_0] - q[w]| + \sum_{i=1}^{d} |s[y_0+i] - q[w+i]| && (3)\\
&> |q[w] - q[x_0]| + \sum_{i=1}^{d} |s[y_0+i] - q[w+i]| && \text{(Equation. 2)}\\
&= \sum_{i=1}^{d} |q[w+i-1] - q[w+i]| + \sum_{i=1}^{d} |s[y_0+i] - q[w+i]| && \text{(Monotonicity of } q\text{)}\\
&= \sum_{i=1}^{d} (|q[w+i-1] - q[w+i]| + |s[y_0+i] - q[w+i]|)\\
&\geq \sum_{i=1}^{d} |q[w+i-1] - s[y_0+i]| && \text{(Triangle Inequality)}\\
&= \mathrm{Cost}(E').
\end{aligned}$$

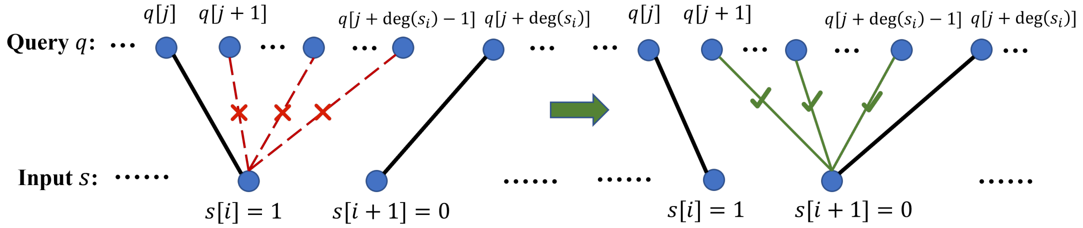

Fig. 5. Obtaining a lower cost matching by shifting matched edges (c.f. Lemma VI.18).

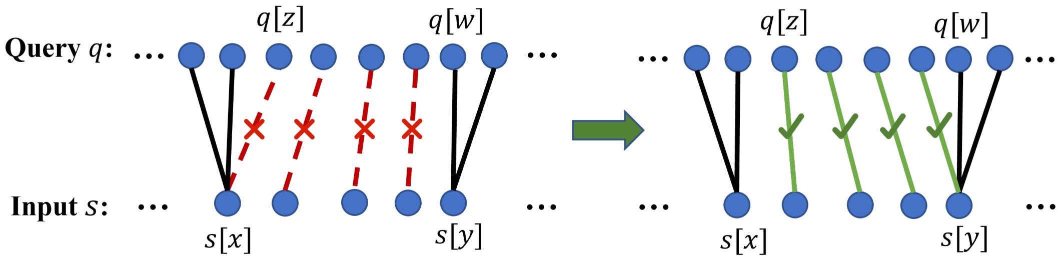

Fig. 6. An illustration of the shifting operation (c.f. Definition VI.2).

$1 \Rightarrow s_j = 0$ in M_{i0}. By Claim VI.21 we know that we can obtain M_{i0} by applying a series of shifting operations to M_i^*, and according to Claim VI.20 we would have $\text{Cost}(M_i^*) \leq \text{Cost}(M_{i0})$. Thus, M_i^* is an optimal matching between $q^{(i)}$ and s. □

Proposition VI.22. *Let $x_j \in \{0,1\}$ be the value of the character matched to $q^{(i)}[j]$ in all isomorphic DTW matchings M_i^*, where $j \in [n]$. We denote the sequence $x := x[1]\dots x[n]$, where $x[j] = x_j$ for $j \in [n]$. The sequence x can be obtained by amplifying the leftmost 0 in s.*

Proof. We see the proposition is naturally true based on the construction of M_i^* in the proof of Lemma VI.19. □

Algorithm to recover DTW matching M_i^*. We now give the algorithm to recover the *isomorphic* DTW matchings M_i^* with the query set $\mathcal{Q}$ (Algorithm 2: line 8-15). The query result d_i of $q^{(i)} = a^{n-i}b^i$ would be $d_i = \sum_{j=1}^{n-i} |x[j]-a| + \sum_{j=n-i+1}^{n} |x[j]-b|$. Recall that $a = \frac{1}{3}$ and $b = \frac{2}{5}$. Consider $q^1 = a^{n-1}b$, where $d_1 = \sum_{j=1}^{n-1} |x[j] - 1/3| + |x[n] - 2/5|$. By computing $(d_1 * 15) \mod 5$, we can know whether $x[n]$ is 0 or 1. For $i > 1$, we have $d_i - d_{i-1} = (\sum_{j=1}^{n-i} |x[j]-a| + \sum_{j=n-i+1}^{n} |x[j]-b|) - (\sum_{j=1}^{n-i+1} |x[j]-a| + \sum_{j=n-i}^{n} |x[j]-b|) = |x[n-i+1]-b| - |x[n-i+1]-a|$. By computing $((d_i - d_{i-1}) * 15) \mod 5$, we can know whether $x[n-i+1]$ is 0 or 1. Then we can recover all $x[j]$'s using this procedure.

Algorithm to recover input sequence s. We now give an overall algorithm (as shown in Algorithm 2) that recovers s using the matching recovery algorithm and claims. For the all 0 and all 1 input sequences, we can use $q^{(n+1)} = 0$, $q^{(n+2)} = 1$ to directly recover them (Algorithm 2: line 2-5). For the rest of the cases, we first recover the optimal isomorphic matching using the described algorithm (Algorithm 2: line 6-15). Let the recovered matching for $\mathcal{Q}$ be $m = x[1]\dots x[n]$, ($x[i] \in \{0,1\}$). Denote the position of the leftmost 0 in x to be u ($1 \leq u \leq \ell$). Then we know $s[1],\dots,s[u-1] = 1$ by Proposition VI.22. By using the sequence $q^{(n+2)} = 1$ to query s, we get the total number of 0's (n_0) in s. Consider the substring $x[u,n]$, and delete the leading zeros in $x[u,n]$ until it has n_0 zeros. Suppose we obtain string s_d after the deletion. We know that $s = 1^{u-1}s_d$ (Algorithm 2: line 16-24). ■

Remark. If we are allowed to use $\mathcal{O}(n)$ extra characters in our queries, we have non-adaptive solutions with $\mathcal{O}(1)$ query complexity for DTW distance. This assumption is stronger than the problem setting (where only $\mathcal{O}(1)$ extra characters are considered) throughout the paper. For details of this complementary result, see Appendix D.

VII. Recovery with Non-Adaptive Fréchet Distance Oracle Queries

Consider two sequences x and y ($x \neq y$) defined on the binary alphabet $\{0,1\}$. The query result from a Fréchet distance oracle only gives very limited information, viz. 0 or 1 (which is more limited than the query from DTW oracle). This 1-bit binary information restricts the power of sequence recovery with Fréchet oracle. Note that it is not possible to distinguish any sequences x and y under Fréchet distance. To see this and to see why the recovery problem is interesting for Fréchet distance, we first define the concept of equivalent sequences *under Fréchet distance* and revisit the problem from the perspective of equivalent sequences.

Definition VII.1 (Equivalent Sequences under Fréchet Distance)**.** Given two sequences x and y, we say x and y are equivalent if y is obtained by taking any bit in x and copying this bit contiguously any number of times. For any pair of equivalent sequences, the Fréchet distance between them is 0.

A simple example of equivalent sequences under Fréchet distance is two sequences, 1 and 11. 11 can be seen as copying the bit 1 in the first sequence and the Fréchet distance between

Algorithm 2: Exact Recovery Algorithm via Queries to DTW Distance Oracle ($\mathcal{O}(1)$ Extra Chars)

```
Input: Non-adaptive query sequences $\mathcal{Q} = \{q^{(1)}, q^{(2)}, \dots, q^{(n+2)}\}$, where $q^{(n+1)} = 0$, $q^{(n+2)} = 1$ and the rest of the
       queries follows our construction;
       The DTW distance query results $\mathcal{R} = \{d_1, d_2, \dots, d_{n+2}\}$ aligned from each query sequence in $\mathcal{Q}$ to the input
       sequence to be recovered.
Output: The sequence $s$ to be recovered.
Function RecoveryDTW ($\mathcal{Q}, \mathcal{R}$):
    if $d_{n+1}$ = 0 then
        return s := $0^{d_{n+2}}$
    if $d_{n+2}$ = 0 then
        return s := $1^{d_{n+1}}$
    positions:= []
    coef_1 := 0
    for $i \in [1, n]$ do                                    ▷ Corresponding queries $q^{(i)} = a^{n-i}b^i$
        coef := $d_i * 15 * 2 \mod 5$
        if (coef − coef_1 + 5) mod 5 = 2 then
            positions.append(0)
        else if (coef − coef_1 + 5) mod 5 = 3 then
            positions.append(1)
        coef_1 := coef
    positions.reverse()
    sequence := [], $i$ := 0
    $n$_0 := $d_{n+2}$, $n$_1 := $d_{n+1}$
    while positions[$i$] = 1 do
        sequence.append(1)
        $i$ += 1
    $i$ += $n - n\_0 - n\_1$
    while $i < n$ do
        sequence.append(positions[$i$])
        $i$ += 1
    return s := sequence
```

1 and 11 is 0. In addition, these two sequences cannot be distinguished by *any* query sequence. This is because for the second sequence, the double 1 characters can be matched to the same character in the query sequence as the single 1 sequence. This will not change the Fréchet distance because the l_∞ norm of the cost of matching edges is not changed.

From the perspective of equivalent sequences, for any two sequences x and y, they are either in the same equivalence class (the Fréchet distance is 0) or in different equivalence classes (the Fréchet distance is 1). Thus the Fréchet distance between two sequences reflects whether or not they are equivalent. Any equivalent sequences, therefore as suggested by its name, are not distinguishable, because all queries from the same equivalence class return 0 and all queries from different equivalence classes return 1. Further, we can categorize all the equivalence classes under Fréchet distance and then derive the lower bound of query complexity of recovering *non-equivalent sequences under Fréchet distance*, which is shown in the following theorem.

Theorem VII.1 (Lower Bound of Recovery from Fréchet Distance)**.** *For a binary alphabet $\{0, 1\}$, any algorithm to recover an arbitrary input sequence $s \in \{0, 1\}^i$ up to equivalence, where $0 \le i \le n$, by querying its Fréchet distance to a non-adaptive set of sequences requires a query complexity of $\Omega(n)$.*

Proof. We begin this proof of query complexity lower bound with a classification of all equivalence classes under the Fréchet distance. For each length $1 \le i \le n$, there exists two non-equivalent sequences under Fréchet distance, which are $\underbrace{010101\dots}_{\text{of length } i}$ and $\underbrace{101010\dots}_{\text{of length } i}$, yielding $2n$ mutually non-equivalent sequences in total. As the Fréchet distance oracle returns 0 when the input sequence and the query sequence are equivalent and 1 otherwise, we would need at least $2n - 1$ queries to exactly recover the input sequence. If the number of queries is less than $2n - 1$, we can always select 2 sequences from the $2n$ mutually non-equivalent sequences which are not covered by the queries, and these two sequences cannot be distinguished by the query sequences. This yields an $\Omega(n)$ lower bound on the query complexity. □

In the analysis of non-adaptive strategies for DTW distance, we have shown that, with extra characters, we can obtain stronger results in recovering the exact sequence. However, using queries from the extended alphabet (no matter how many extra characters are allowed) does not help increase the power of recovery under Fréchet distance, proved in the following theorem.

Theorem VII.2 (Extra Characters Are Not Helpful)**.** *Given two sequences s and s', if the Fréchet distance $d_F(s, s') = 0$, then any query q with extra characters cannot distinguish s and s'.*

Proof. Given sequences s, s' (where $d_F(s, s') = 0$) and query q with extra characters, our goal is to show $d_F(s, q) = d_F(s', q)$ for *every possible* q. The technique of our proof is, for an optimal matching between s and any query q, we can construct a matching between s' and q with the same cost, and vice versa. In this way, we know that $d_F(s', q) \le d_F(s, q)$ and $d_F(s, q) \le d_F(s', q)$, so $d_F(s, q) = d_F(s', q)$ and q cannot distinguish s and s'.

Since $d_F(s, s') = 0$, s and s' have the same condensed expression. Suppose s and s' has k runs. In the optimal matching between q and s, let $q_{s^{(i)}}$ denote the substring in q which is matched to the i-th run of s for every $i \in [k]$. We can always match all $q_{s^{(i)}}$'s to the i-th run of s' instead. Note that the i-th runs of s and s' (denoted by $s^{(i)}$ and $s'^{(i)}$, resp.) are of the same character with maybe various length. The Fréchet distance between $q_{s^{(i)}}$ and $s^{(i)}$ only depends on the characters in $q_{s^{(i)}}$ and thus $d_F(q_{s^{(i)}}, s^{(i)}) = d_F(q_{s^{(i)}}, s'^{(i)})$. Therefore we obtain a matching between s' and q with a cost of $d_F(s, q)$. This matching between q and s' may be not optimal but is valid, and therefore we can conclude $d_F(s', q) \le d_F(s, q)$. Due to the symmetry of the statement, we can similarly obtain $d_F(s, q) \le d_F(s', q)$. This finishes the proof of this theorem. □

Since extra characters are not helpful in recovering from Fréchet distance queries, we conclude the analysis with a *trivially interesting* approach to recover sequences up to equivalence. The approach uses up to $2n - 1$ queries, which exactly matches our query complexity lower bound, as shown in the following theorem.

Theorem VII.3 (Non-adaptive Strategy for Fréchet Equivalence Class Recovery)**.** *For a binary alphabet $\{0, 1\}$ and two input sequences $s, s' \in \{0, 1\}^i$ where $0 \le i \le n$ and s and s' are non-equivalent sequences under Fréchet distance, there exists an algorithm to distinguish the input sequences s and s', given $2n - 1 \in \mathcal{O}(n)$ query sequences $\mathcal{Q}$ and the Fréchet distance of s and s' to each query sequence $q \in \mathcal{Q}$.*

Proof. We first show that, for each length $0 \le i \le n$, there are only two non-equivalent sequences under Fréchet distance, which are 010101... and 101010... sequences, viz., we can identify two non-equivalent sequences by specifying the sequence length i and the starting bit. Therefore, for the maximum sequence length n, there are only $2n$ mutually non-equivalent sequences.

Given any two different sequences from this $2n$-sized collection of non-equivalent sequences under Fréchet distance, we can use $\mathcal{O}(n)$ query sequences to distinguish them. That is, we can utilize the exact set of $2n$ non-equivalent sequences as the query sequences. If the query sequence p is exactly the input sequence q, the Fréchet distance between p and q is $d_F(p, q) = 0$. If the query sequence p is not equivalent to the input sequence q, then the Fréchet distance between p and q is $d_F(p, q) = 1$ because it is impossible to skip over a bit without paying cost 1. Note that any one of the $2n$ queries can be skipped since we know the fact that there would be exactly one 0 among the $2n$ query results. Therefore, $2n - 1 \in \mathcal{O}(n)$ query sequences suffice to distinguish any two sequences from the non-equivalent sequence set and this finishes the proof. □

This theorem shows that, if an input is in the collection of non-equivalent sequences under Fréchet distance, we can use $\mathcal{O}(n)$ queries to exactly recover this sequence given the query results under the Fréchet distance.

Remark: Extension to non-binary alphabets. Our results are presented for input sequences from binary alphabet $\{0, 1\}$. These results can be extended to any non-binary alphabet Σ by encoding the non-binary alphabet in a binary domain. This will increase the query complexity by a constant factor from $|\Sigma|$ (one-hot encoding) to $\log(|\Sigma|)$ (binary encoding). This extension works for the results for all distance metrics shown in this paper. However, we note that this extension may not be optimal if one considers a large alphabet (e.g., larger than n). In fact, calculating some of the distances themselves on a general alphabet is under SETH [28], [29], which is a much hard problem than on the binary case [30]. Obtaining optimal results on the extension of the non-decomposable distance recovery problem leaves room for future research.

VIII. Related Work

A distance embedding [31] embeds sequences from the original distance metric space to other distance measures (usually l_p norms), such that the distance measurements in the original space can be preserved up to a factor of D, namely *the distortion rate*. The sequence distance embedding problem is related to our problem in the sense that, in our problem, we intend to recover the input sequence from a list of query results that are in the l_p space, which can be regarded as finding a special distance embedding. Existing works on the sequence distance embedding problem mainly focus on constructing such an embedding which can have a close approximation (viz., *low distortion rate*) and reduce the computational complexity (i.e., cost) on the new distance space. [32] shows a lower bound of $3/2$ on the distortion rate of embedding edit distance into ℓ_p norm spaces. An improvement of $(\log n)^{\frac{1}{2} - o(1)}$ on this lower bound [33] has been further simplified and improved into $\Omega(\log n)$ by [34].

Distance embeddings can be used to estimate the distance on the complex metric space because the evaluation and computations on the new (simpler metric) space can be significantly faster [31]. Under the asymmetric query model (when estimating the edit distance between x and y, the algorithm has unrestricted power accessing x but limited power accessing y), [35] proposes a $(\log n)^{\mathcal{O}(1/\epsilon)}$ approximation algorithm that runs in $n^{1+\epsilon}$ time. [36] considers the alignment problem when estimating the edit distance (finding the sequence of edits between the estimated sequences) and presents an alignment with $(\log n)^{\mathcal{O}(1/\epsilon^2)}$ approximation in time $\tilde{\mathcal{O}}(n^{1+\epsilon})$. The sequence distance embedding problem has been investigated on other distance metrics as well, for example, the block edit distance [31] and the Ulam distance [37]. Existing work also shows embeddings from edit distance to the Hamming space [38], [39]. However, to the best of our knowledge, there is no prior work considering the embedding problem of the *DTW distance* and the *exact* recovery problem based on distance oracle query results.

IX. Open Problems

We initiate an exact recovery problem of sequences using queries to a non-decomposable distance oracle. We show recovery algorithms for edit distance, DTW distance, and Fréchet distance, as well as a general adaptive algorithm for a wide class of distance oracles. We envision the following directions for future work.

First, for the edit distance, there is still a quadratic gap between the non-adaptive query complexity upper and lower bounds without extra characters. Closing this gap requires a deeper understanding about the properties of edit distance.

Second, for the DTW distance, it remains unclear whether 1 extra character suffices for an $\mathcal{O}(n)$ non-adaptive upper bound, or we can have an $\Omega(n^2)$ non-adaptive lower bound with 1 extra character (our proof uses 2 extra characters).

Furthermore, as the initial work on non-decomposable distance recovery problem, we consider a simpler setting where input sequences are drawn from binary alphabet $\{0, 1\}$. While our results can be naturally extended to a non-binary alphabet, as stated in the paper, with a compensation of increasing the query complexity up to a constant factor, we notice that for some distances (e.g., DTW), the calculation on the general alphabet is much harder than on the binary case. This spawns the open question for follow-up work to consider: Would there exist a strategy specifically designed for the non-binary alphabet with lower query complexity (than using encoding extensions to our results on the binary alphabet)?

Lastly, it would be interesting to consider the exact sequence recovery problem using the properties of specific distance metrics. For example, the Edit distance with Real Penalty (ERP) distance [40] which supports local time shifting in time series by the marriage of the ℓ_1 norm and edit distance, would be of interest. One can also consider other variants of our problem in terms of adaptive queries or the approximate recovery problem in the presence of noise.

Acknowledgment

The authors would like to thank the anonymous reviewers of ITCS 2023 and IEEE Transactions of Information Theory for their detailed and constructive comments which helped to improve the paper vastly during the revision process. David P. Woodruff would like to thank support from ONR grant N00014-18-1-2562 and a Simons Investigator Award. Hongyang Zhang would like to thank support from NSERC Discovery Grant RGPIN-2022-03215, DGECR-2022-00357.

References

[1] Z. Hu, X. Li, D. P. Woodruff, H. Zhang, and S. Zhang, "Recovery from Non-Decomposable Distance Oracles," in *14th Innovations in Theoretical Computer Science Conference (ITCS 2023)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), Y. Tauman Kalai, Ed., vol. 251. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, pp. 73:1–73:22. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2023/17576

[2] H. S. Shapiro and N. Fine, "E1399," *The American Mathematical Monthly*, vol. 67, no. 7, pp. 697–698, 1960.

[3] N. H. Bshouty, "Optimal algorithms for the coin weighing problem with a spring scale," in *COLT 2009 - The 22nd Conference on Learning Theory, Montreal, Quebec, Canada, June 18-21, 2009*, 2009. [Online]. Available: http://www.cs.mcgill.ca/%7Ecolt2009/papers/004.pdf#page=1

[4] R. Dorfman, "The detection of defective members of large populations," *The Annals of Mathematical Statistics*, vol. 14, no. 4, pp. 436–440, 1943.

[5] M. Aldridge, O. Johnson, and J. Scarlett, "Group testing: An information theory perspective," *Found. Trends Commun. Inf. Theory*, vol. 15, no. 3-4, pp. 196–392, 2019. [Online]. Available: https://doi.org/10.1561/0100000099

[6] A. Coja-Oghlan, O. Gebhard, M. Hahn-Klimroth, and P. Loick, "Optimal group testing," in *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*, ser. Proceedings of Machine Learning Research, J. D. Abernethy and S. Agarwal, Eds., vol. 125. PMLR, 2020, pp. 1374–1388. [Online]. Available: http://proceedings.mlr.press/v125/coja-oghlan20a.html

[7] C. Wang, Q. Zhao, and C. Chuah, "Optimal nested test plan for combinatorial quantitative group testing," *IEEE Trans. Signal Process.*, vol. 66, no. 4, pp. 992–1006, 2018. [Online]. Available: https://doi.org/10.1109/TSP.2017.2780053

[8] S. Soderberg and H. S. Shapiro, "A combinatory detection problem," *The American Mathematical Monthly*, vol. 70, no. 10, pp. 1066–1070, 1963. [Online]. Available: http://www.jstor.org/stable/2312835

[9] D. G. Cantor and W. H. Mills, "Determination of a subset from certain combinatorial properties," *Canadian Journal of Mathematics*, vol. 18, p. 42–48, 1966.

[10] M. Li and P. M. B. Vitányi, "Combinatorics and kolmogorov complexity," in *Proceedings of the Sixth Annual Structure in Complexity Theory Conference, Chicago, Illinois, USA, June 30 - July 3, 1991*. IEEE Computer Society, 1991, pp. 154–163. [Online]. Available: https://doi.org/10.1109/SCT.1991.160256

[11] D. E. Knuth, "The computer as Master Mind," *Journal of Recreational Mathematics*, vol. 9, no. 1, pp. 1–6, 1976.

[12] P. Afshani, M. Agrawal, B. Doerr, C. Doerr, K. G. Larsen, and K. Mehlhorn, "The query complexity of a permutation-based variant of mastermind," *Discret. Appl. Math.*, vol. 260, pp. 28–50, 2019. [Online]. Available: https://doi.org/10.1016/j.dam.2019.01.007

[13] M. Fernandez, D. P. Woodruff, and T. Yasuda, "The query complexity of mastermind with l_p distances," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2019, September 20-22, 2019, Massachusetts Institute of Technology, Cambridge, MA, USA*, ser. LIPIcs, D. Achlioptas and L. A. Végh, Eds., vol. 145. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 1:1–1:11. [Online]. Available: https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2019.1

[14] J. A. Rodríguez-Velázquez, I. G. Yero, D. Kuziak, and O. R. Oellermann, "On the strong metric dimension of Cartesian and direct products of graphs," *Discret. Math.*, vol. 335, pp. 8–19, 2014. [Online]. Available: https://doi.org/10.1016/j.disc.2014.06.023

[15] Z. Jiang and N. Polyanskii, "On the metric dimension of cartesian powers of a graph," *J. Comb. Theory, Ser. A*, vol. 165, pp. 1–14, 2019. [Online]. Available: https://doi.org/10.1016/j.jcta.2019.01.002

[16] M. Lécuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 656–672. [Online]. Available: https://doi.org/10.1109/SP.2019.00044

[17] J. M. Cohen, E. Rosenfeld, and J. Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 1310–1320. [Online]. Available: http://proceedings.mlr.press/v97/cohen19c.html

[18] X. Cai, T. Xu, J. Yi, J. Huang, and S. Rajasekaran, "DTWNet: a dynamic time warping network," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 11 636–11 646. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/02f063c236c7eef66324b432b748d15d-Abstract.html

[19] R. Vershynin, "Lectures in geometric functional analysis," *Unpublished manuscript. Available at http://www-personal. umich. edu/romanv/papers/GFA-book/GFA-book. pdf*, vol. 3, no. 3, pp. 3–3, 2011.

[20] M. Buchin, A. Driemel, K. van Greevenbroek, I. Psarros, and D. Rohde, "Approximating length-restricted means under dynamic time warping," in *Approximation and Online Algorithms - 20th International Workshop, WAOA 2022, Potsdam, Germany, September 8-9, 2022, Proceedings*, ser. Lecture Notes in Computer Science, P. Chalermsook and B. Laekhanukit, Eds., vol. 13538. Springer, 2022, pp. 225–253. [Online]. Available: https://doi.org/10.1007/978-3-031-18367-6_12

[21] V. Braverman, M. Charikar, W. Kuszmaul, D. P. Woodruff, and L. F. Yang, "The one-way communication complexity of dynamic time warping distance," in *35th International Symposium on Computational Geometry, SoCG 2019, June 18-21, 2019, Portland, Oregon, USA*, ser. LIPIcs, G. Barequet and Y. Wang, Eds., vol. 129. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 16:1–16:15. [Online]. Available: https://doi.org/10.4230/LIPIcs.SoCG.2019.16

[22] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, 1966, pp. 707–710.

[23] T. Eiter and H. Mannila, "Computing discrete fréchet distance," Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, Tech. Rep. CD-TR 94/64, April 1994. [Online]. Available: http://www.kr.tuwien.ac.at/staff/eiter/et-archive/cdtr9464.pdf

[24] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk, "Fréchet distance for curves, revisited," in *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, ser. Lecture Notes in Computer Science, Y. Azar and T. Erlebach, Eds., vol. 4168. Springer, 2006, pp. 52–63. [Online]. Available: https://doi.org/10.1007/11841036_8

[25] A. Abboud, A. Backurs, and V. V. Williams, "Tight hardness results for LCS and other sequence similarity measures," in *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, V. Guruswami, Ed. IEEE Computer Society, 2015, pp. 59–78. [Online]. Available: https://doi.org/10.1109/FOCS.2015.14

[26] N. Schaar, V. Froese, and R. Niedermeier, "Faster binary mean computation under dynamic time warping," in *31st Annual Symposium on Combinatorial Pattern Matching, CPM 2020, June 17-19, 2020, Copenhagen, Denmark*, ser. LIPIcs, I. L. Gørtz and O. Weimann, Eds., vol. 161. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 28:1–28:13. [Online]. Available: https://doi.org/10.4230/LIPIcs.CPM.2020.28

[27] I. Kremer, N. Nisan, and D. Ron, "On randomized one-round communication complexity," in *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, F. T. Leighton and A. Borodin, Eds. ACM, 1995, pp. 596–605. [Online]. Available: https://doi.org/10.1145/225058.225277

[28] A. Abboud, T. D. Hansen, V. V. Williams, and R. Williams, "Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, D. Wichs and Y. Mansour, Eds. ACM, 2016, pp. 375–388. [Online]. Available: https://doi.org/10.1145/2897518.2897653

[29] K. Bringmann and M. Künnemann, "Quadratic conditional lower bounds for string problems and dynamic time warping," in *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, V. Guruswami, Ed. IEEE Computer Society, 2015, pp. 79–97. [Online]. Available: https://doi.org/10.1109/FOCS.2015.15

[30] W. Kuszmaul, "Binary dynamic time warping in linear time," *CoRR*, vol. abs/2101.01108, 2021. [Online]. Available: https://arxiv.org/abs/2101.01108

[31] G. Cormode, "Sequence distance embeddings," Ph.D. dissertation, University of Warwick, Coventry, UK, 2003. [Online]. Available: http://wrap.warwick.ac.uk/61310/

[32] A. Andoni, M. Deza, A. Gupta, P. Indyk, and S. Raskhodnikova, "Lower bounds for embedding edit distance into normed spaces," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*. ACM/SIAM, 2003, pp. 523–526. [Online]. Available: http://dl.acm.org/citation.cfm?id=644108.644196

[33] S. Khot and A. Naor, "Nonembeddability theorems via fourier analysis," in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*. IEEE Computer Society, 2005, pp. 101–112. [Online]. Available: https://doi.org/10.1109/SFCS.2005.54

[34] R. Krauthgamer and Y. Rabani, "Improved lower bounds for embeddings intol$_{1}$\$," *SIAM J. Comput.*, vol. 38, no. 6, pp. 2487–2498, 2009. [Online]. Available: https://doi.org/10.1137/060660126

[35] A. Andoni, R. Krauthgamer, and K. Onak, "Polylogarithmic approximation for edit distance and the asymmetric query complexity," in *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*. IEEE Computer Society, 2010, pp. 377–386. [Online]. Available: https://doi.org/10.1109/FOCS.2010.43

[36] M. Charikar, O. Geri, M. P. Kim, and W. Kuszmaul, "On estimating edit distance: Alignment, dimension reduction, and embeddings," in *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, ser. LIPIcs, I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, Eds., vol. 107. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, pp. 34:1–34:14. [Online]. Available: https://doi.org/10.4230/LIPIcs.ICALP.2018.34

[37] M. Charikar and R. Krauthgamer, "Embedding the ulam metric into l_1," *Theory Comput.*, vol. 2, no. 11, pp. 207–224, 2006. [Online]. Available: https://doi.org/10.4086/toc.2006.v002a011

[38] D. Belazzougui and Q. Zhang, "Edit distance: Sketching, streaming, and document exchange," in *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, I. Dinur, Ed. IEEE Computer Society, 2016, pp. 51–60. [Online]. Available: https://doi.org/10.1109/FOCS.2016.15

[39] D. Chakraborty, E. Goldenberg, and M. Koucký, "Streaming algorithms for embedding and computing edit distance in the low distance regime," in *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, D. Wichs and Y. Mansour, Eds. ACM, 2016, pp. 712–725. [Online]. Available: https://doi.org/10.1145/2897518.2897577

[40] L. Chen and R. T. Ng, "On the marriage of l_p-norms and edit distance," in *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004*, M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, Eds. Morgan Kaufmann, 2004, pp. 792–803. [Online]. Available: http://www.vldb.org/conf/2004/RS21P2.PDF

[41] A. Amir, M. Amit, G. M. Landau, and D. Sokol, "Period recovery of strings over the hamming and edit distances," *Theor. Comput. Sci.*, vol. 710, pp. 2–18, 2018. [Online]. Available: https://doi.org/10.1016/j.tcs.2017.10.026

[42] J. Sima and J. Bruck, "Trace reconstruction with bounded edit distance," in *IEEE International Symposium on Information Theory, ISIT 2021, Melbourne, Australia, July 12-20, 2021*. IEEE, 2021, pp. 2519–2524. [Online]. Available: https://doi.org/10.1109/ISIT45174.2021.9518244

[43] M. Bressan, N. Cesa-Bianchi, S. Lattanzi, and A. Paudice, "Exact recovery of clusters in finite metric spaces using oracle queries," in *Conference on Learning Theory, COLT 2021, 15-19 August 2021, Boulder, Colorado, USA*, ser. Proceedings of Machine Learning Research, M. Belkin and S. Kpotufe, Eds., vol. 134. PMLR, 2021, pp. 775–803. [Online]. Available: http://proceedings.mlr.press/v134/bressan21a.html

[44] A. F. Sunjaya and A. P. Sunjaya, "Pooled testing for expanding covid-19 mass surveillance," *Disaster Medicine and Public Health Preparedness*, vol. 14, no. 3, pp. e42–e43, 2020.

[45] I. Yelin, N. Aharony, E. S. Tamar, A. Argoetti, E. Messer, D. Berenbaum, E. Shafran, A. Kuzli, N. Gandali, O. Shkedi, T. Hashimshony, Y. Mandel-Gutfreund, M. Halberthal, Y. Geffen, M. Szwarcwort-Cohen, and R. Kishony, "Evaluation of COVID-19 RT-qPCR Test in Multi sample Pools," *Clinical Infectious Diseases*, vol. 71, no. 16, pp. 2073–2078, 05 2020. [Online]. Available: https://doi.org/10.1093/cid/ciaa531

[46] M. L. Fredman and D. E. Willard, "BLASTING through the information theoretic barrier with FUSION TREES," in *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, H. Ortiz, Ed. ACM, 1990, pp. 1–7. [Online]. Available: https://doi.org/10.1145/100216.100217

[47] A. Selberg, "An elementary proof of the prime-number theorem," *Annals*

of Mathematics, vol. 50, no. 2, pp. 305–313, 1949. [Online]. Available: http://www.jstor.org/stable/1969455

Appendix A
Other Related Work

Recovery problems in metric spaces. Our problem is related to the recovery or reconstruction problems over metric spaces. [41] study the *period recovery problem* on strings, which is to find the primitive periods between two strings such that the periodic distance is below a threshold. They present an $\mathcal{O}(n \log n)$-time algorithm for Hamming distance and an $\mathcal{O}(n^{4/3})$-time algorithm for edit distance. [42] investigate the approximate recovery problem over bounded edit distance spaces in the presence of noise and show $n^{\mathcal{O}(k)}$ noisy samples suffice for (approximate) reconstruction. Interestingly, [43] consider the *exact* recovery problem using oracle queries but the objective of their work is to exactly recover the clusters in Euclidean space, which is similar but orthogonal to our problem.

Learning problems: Coin-weighing and group-testing problems. The related "decomposable" instance to our problem of querying a Hamming distance oracle is equivalent to the coin-weighing problem [3] and the quantitative group testing problem [7]. Both the coin-weighing problem and group-testing problems are well-studied learning problems in the literature and have many real-world applications [8], [9], [10], [44], [45]. The coin-weighing problem is to determine the weight of each coin (of two distinct weights w_1 and w_2) by using a minimal number of weighings of a subset of n total coins each time. [9] and [3] respectively present $2n/\log n$ weighing solutions which are optimal non-adaptive solutions to this problem. Assuming the number of w_1 weight coins is known to be d, this d-coin weighing problem can be solved by an adaptive algorithm in time $2d \log \frac{n}{d} / \log d + \mathcal{O}(d/\log d + d(\log\log d)\log\frac{n}{d}/(\log d)^2)$ [3]. The major difference between our problem and these well-studied problems is that we consider distance metrics which cannot be aligned and represented as $\sum_i^n f(x_i - y_i)$ (i.e., the edit distance, DTW distance, and Fréchet distances).

Appendix B
Coordinate Descent Algorithm Instantiation

Now we briefly discuss how we apply the Coordinate Descent algorithm to all three distances we consider in this paper by justifying the two conditions hold.

Edit distance. For condition 2, we know that $\forall s, q, \mathrm{dist}(s, q) \leq n$ since the maximum length of s or q is n. For condition 1, in each iteration, we consider a set Q that contains all sequences that can be transformed from q by inserting, deleting or substituting one character in q (edit operations). Note that $|Q|$ cannot exceed $(n + 1) + n + n = 3n + 1$. We claim that there exists a q' in Q such that $\mathrm{dist}(s, q) > \mathrm{dist}(s, q')$. Let $\mathrm{dist}(s, q) = d$. By the definition of edit distance, there exists a chain of edit operations of length d that transforms s to q, resulting in a list of intermediate sequences $q_1, ..., q_{d-1}$. Note that $\mathrm{dist}(s, q) \geq \mathrm{dist}(q_1, q) + 1$, otherwise we have $\mathrm{dist}(q_1, q) > d - 1$. However, the chain implies we can transform q_1 to q in $d - 1$ edit operations, which leads to a contradiction. Since $q_1 \in Q$, we can find q_1 satisfying the condition in $3n + 1$ searches. Therefore, the algorithm is guaranteed to recover the input in $\mathcal{O}(n^2)$ steps.

DTW distance. For DTW distance, condition 2 holds since $\forall s, q, \mathrm{dist}(s, q) \leq n$. For condition 1, consider the $\#\textsc{runs}(x)$ in s and q. If $\#\textsc{runs}(x)$ of $q < s$, then either adding an (arbitrary length) run to the start or the end of q will decrease the DTW distance from s. On the other hand, if $\#\textsc{runs}(x)$ of $q > s$, then either deleting a run from the start or the end of q will decrease the DTW distance from s. If $\#\textsc{runs}(x)$ of $q = s$ and $\mathrm{dist}(s, q) \neq 0$, we can still decrease the distance from q by either adding/deleting a run to the start/end of the sequence. Therefore, the algorithm is guaranteed to recover the input in $\mathcal{O}(n^2)$ steps.

Fréchet distance. Condition 2 holds since $\forall s, q, \mathrm{dist}(s, q) \leq 1$. For condition 1, enumerating $2n$ non-equivalent sequences, (i.e., 010101... and 101010...) guarantees to find q' such that $\mathrm{dist}(s, q) > \mathrm{dist}(s, q') = 0$. Therefore, the algorithm terminates in $\mathcal{O}(n)$ steps.

Appendix C
Proofs of Claim VI.17 in Lemma VI.15

Proof of Claim VI.17. We prove this by contradiction. Suppose $\exists (i, j)$ where $i \in [n]$ and $j \in [\ell]$ such that $\deg(q[i]) > 1$ and $\deg(s[j]) > 1$. Let $\mathbf{X} = \{x \in [n] \mid \deg(q[x]) > 1\}$, $\mathbf{Y} = \{y \in [\ell] \mid \deg(s[y]) > 1\}$ and $\mathbf{Z} = \{z \in [\ell] \mid \exists x \in \mathbf{X}$ such that edge $(q[x], s[z]) \in M\}$. According to Claim VI.16, we know that $\mathbf{Y} \bigcap \mathbf{Z} = \emptyset$. Let $d = \min_{y \in \mathbf{Y}, z \in \mathbf{Z}} |y - z|$. We would have $d > 0$. Suppose we have $x_0 \in \mathbf{X}, y_0 \in \mathbf{Y}, z_0 \in \mathbf{Z}$ such that edge $(q[x_0], s[z_0]) \in M$ and $|y_0 - z_0| = d$. There are two cases to discuss. As we have already solved the case $y_0 < z_0$ in the proofs of Claim VI.17 in Lemma VI.15, here we only discuss the case $y_0 > z_0$.

In this case, we can assume that $s[y_0]$ is matched to

$$\{q[w], q[w + 1], ..., q[w + \deg(s[y_0]) - 1]\}$$

and $q[x_0]$ is matched to

$$\{s[z_0 - \deg(q[x_0]) + 1], s[z_0 - \deg(q[x_0]) + 2], ..., s[z_0]\}.$$

We remove $d{+}1$ edges $E = \{(q[x_0], s[z_0]), (q[x_0{+}1], s[z_0{+}1]), ..., (q[w], s[y_0])\}$and add d new edges

$$E' = \{(q[x_0{+}1], s[z_0]), (q[x_0{+}2], s[z_0{+}1]), ..., (q[w], s[y_0{-}1])\}$$

to construct a new matching M'. Since $\deg(s[y_0]) > 1$ and $\deg(q[x_0]) > 1$ in M, M' would still be a valid matching. Computing the sum of two sets of edges E and E', respectively, would yield Equation. 4 (see the cross-column equations).

Hence, M' would be a better matching than M, a contradiction. Combining 1) and 2) completes the proof of Claim VI.17.

Appendix D
Recovery Using Non-Adaptive DTW Distance Oracle with $\mathcal{O}(n)$ Extra Characters

Theorem D.1 (Non-adaptive Strategy for DTW Exact Recovery with $\mathcal{O}(n)$ Extra Characters)**.** *Define a sequence of* n *elements,*

$$\begin{aligned}
\mathrm{Cost}(E) &= |s[y_0] - q[w]| + \sum_{i=1}^{d} |s[z_0+i-1] - q[x_0+i-1]| && (4)\\
&> |q[w] - q[x_0]| + \sum_{i=1}^{d} |s[z_0+i-1] - q[x_0+i-1]| && \text{(Equation. 2)}\\
&= \sum_{i=1}^{d} |q[x_0+i-1] - q[x_0+i]| + \sum_{i=1}^{d} |s[z_0+i-1] - q[x_0+i-1]| && \text{(Monotonicity of } q\text{)}\\
&= \sum_{i=1}^{d} (|q[x_0+i-1] - q[x_0+i]| + |s[z_0+i-1] - q[x_0+i-1]|) &&\\
&\geq \sum_{i=1}^{d} |q[x_0+i] - s[z_0+i-1]| && \text{(Triangle Inequality)}\\
&= \mathrm{Cost}(E').
\end{aligned}$$

each of which has $\mathcal{O}(\log n)$ bit complexity, as a query sequence. For a binary alphabet $\{0,1\}$ and an input sequence $s := \{0,1\}^\ell$ where $0 \leq \ell \leq n$, there exists an algorithm to recover the input sequence s, given $4 \in \mathcal{O}(1)$ query sequences $\mathcal{Q}$ and the $d_{\mathrm{DTW}}(s,q)$ to each query sequence $q \in \mathcal{Q}$.

We note that, if we remove the constraint of $\mathcal{O}(\log n)$ bit complexity, we can give a straightforward solution by leveraging the query string $q = \{\frac{1}{(n+1)}, \frac{1}{(n+1)^2}, \dots, \frac{1}{(n+1)^n}\}$ to encode much more information in a single query. With the word RAM bit complexity requirement [46] on the queries though, namely that each entry fits into a single $O(\log n)$-bit word, such solutions are not allowed.

Proof of Theorem D.1. Note that we can still use query sequences 0 and 1 to recover input sequences consisting of only 0s or 1s. For simplicity, we assume in the rest of the proof that the input sequence s contains both 0 and 1 and let $s = s[1]s[2]...s[\ell]$.

We give our proof by constructing 2 query sequences q and q' and presenting an algorithm to recover an input sequence s from its DTW distance to these 2 query sequences.

Query Sequences Construction. Let $P_{prime} = \{p_1, p_2, ..., p_n\}$ be the first n primes not including 2. By the prime number theorem [47], we have that $p_n = \mathcal{O}(n \log n)$. Note that for any prime number $p_i > 2$, $\exists 1 \leq x_i < p_i$ such that $\frac{1}{4} < \frac{x_i}{p_i} < \frac{1}{2}$. We obtain q by selecting such a $\frac{x_i}{p_i}$ for each $p_i \in P_{prime}$ and rearranging them in increasing order. Then we construct q as $q = q[1]q[2]...q[n]$ where $\frac{1}{4} < q[1] < q[2] < ... < q[n] < \frac{1}{2}$. Let $q'[i] = 1 - q[i], 1 \leq i \leq n$, and let $q' = q'[1]q'[2]...q'[n]$. We would have $\frac{3}{4} > q'[1] > q'[2] > ... > q'[n] > \frac{1}{2}$. Since $p_n = \mathcal{O}(n \log n)$, it is easy to verify that each $q[i]$ and $q'[i]$ does have bit complexity $\mathcal{O}(\log n)$.

According to Lemma VI.15, each element in q would be involved exactly once in $d_{\mathrm{DTW}}(q,s)$, and a similar argument would hold for q'. We hereby present an algorithm to determine the value of the matched element for each element in q, and the same algorithm can also be applied to q'.

Algorithm to determine matched elements for a query sequence. Suppose $q[i] = \frac{x_{t_i}}{p_{t_i}}$ where $\{t_j\}$ is a permutation of $[n]$. Letting m_i be the value matched to $q[i]$ in the optimal DTW matching for (q,s) (different m_is could correspond to the same element in s), $m_i \in \{0,1\}$, we would have

$$d_{\mathrm{DTW}}(q,s) = \sum_{i=1}^{n} |m_i - q[i]| = \sum_{i=1}^{n} \left| \frac{m_i p_{t_i} - x_{t_i}}{p_{t_i}} \right| = \frac{\sum_{i=1}^{n} \left(|m_i p_{t_i} - x_{t_i}| \cdot \Pi_{j \neq t_i} p_j \right)}{\Pi_{i=1}^{n} p_i}.$$

Let $d_{\mathrm{DTW}}(q,s) = \frac{u}{v}$, where u and v are co-primes. We have $u = \sum_{i=1}^{n} \left(|m_i p_{t_i} - x_{t_i}| \cdot \Pi_{j \neq t_i} p_j \right)$ and $v = \Pi_{i=1}^{n} p_i$. Consider $u \bmod p_{t_k}$ for a specific k. As each term in the summation has a factor p_{t_k} except $|m_k p_{t_k} - x_{t_k}| \cdot \Pi_{j \neq t_k} p_j$, we have $a \equiv |m_k p_{t_k} - x_{t_k}| \cdot \Pi_{j \neq t_k} p_j \mod p_{t_k}$. Note that $p_{t_k} - x_{t_k} \not\equiv x_{t_k} \mod p_{t_k}$, so $(p_{t_k} - x_{t_k}) \cdot \Pi_{j \neq t_k} p_j \not\equiv x_{t_k} \cdot \Pi_{j \neq t_k} p_j \bmod p_{t_k}$. Thus, we can determine m_k by checking whether $(p_{t_k} - x_{t_k}) \cdot \Pi_{j \neq t_k} p_j \equiv u \mod p_{t_k}$ or $x_{t_k} \cdot \Pi_{j \neq t_k} p_j \equiv u \mod p_{t_k}$.

Furthermore, we have the following claim for the optimal DTW matching between q, q' and s.

Claim D.2. For any given input sequence s and optimal DTW matching M and M' for (q,s) and (q',s) respectively, we have $\deg(s_[i]) = 1$ in M if $s[i] = 1$ and $\deg(s[i]) = 1$ in M' if $s[i] = 0$.

Proof of claim. We give a proof by contradiction. Given an optimal DTW matching M for (q,s), suppose $\exists 1 \leq i \leq \ell$ such that $s[i] = 1$ and $\deg(s[i]) > 1$. Suppose $s[i]$ is matched to $q[j], q[j+1], \dots, q[j + \deg(s[i]) - 1]$.

First, we show that we can "swap" $s[i]$ with its neighboring element while maintaining optimality of the matching. If one of the neighboring elements of $s[i]$ is 1, without loss of generality, suppose $s[i+1] = 1$, then we can construct an alternate optimal matching M^* where $\deg(s[i]) = 1$ and $\deg(s[i+1]) > 1$. According to Lemma VI.15, $s[i+1]$ cannot be matched with any of $q[j], q[j+1], \dots, q[j+\deg(s[i])-1]$ in M, otherwise there would exist $j+1 \leq k \leq j+\deg(s[i])-1$ such that $\deg(q[k]) = 2$. Thus by matching $q[j+1], \dots, q[j+\deg(s[i])-1]$ to $s[i+1]$ instead of $s[i]$, we would obtain a new optimal matching M^*

where $\deg(s[i]) = 1$ and $\deg(s[i+1]) > 1$.

As there exists at least one 0 in s, we know that there exists an optimal DTW matching M_0^* for (q, s) where $\exists s[i]$ such that $s[i] = 1$, $\deg(s[i]) > 1$ and one of the neighboring elements of $s[i]$ is 0. Without loss of generality, suppose $s[i+1] = 0$. Similarly, according to Lemma VI.15, $s[i+1]$ cannot be matched with any of $q[j], q[j+1], \ldots, q[j + \deg(s[i]) - 1]$ in M_0. Here we construct a new matching M_0' by matching $q[j+1], \ldots, q[j + \deg(s[i]) - 1]$ to $s[i+1]$ instead of $s[i]$. Fig 5 illustrates an example of such a construction. Considering the total cost of differing edges in both matchings, we have $\sum_{k=j+1}^{j+\deg(s[i])-1} |s[i] - q[k]| > \sum_{k=j+1}^{j+\deg(s[i])-1} \frac{1}{2} > \sum_{k=j+1}^{j+\deg(s[i])-1} |s[i+1] - q[k]|$. Thus M_0^* would be a better matching than M_0, causing a contradiction and thus finishing the proof. A similar proof can be derived for query sequence q' and the case $s[i] = 0$.

Algorithm to recover s. We now give an overall algorithm that recovers s using the above algorithm and claim. Applying the above algorithm gives the matched elements of q and q'. Let the matching result for q and q' be $m = m_1 ... m_n$ and $m' = m_1' ... m_n'$, $(m_i, m_i' \in \{0, 1\})$ respectively. We break m and m' into blocks such that each block is the longest substring that contains either 0 or 1. By also breaking s into such blocks, we know that m has the same number of blocks as s according to Lemma VI.15. Similarly m' has the same number of blocks as s. Let l be the number of blocks that m and m' have. Then we can represent m and m' as $m = A_1 ... A_l$ and $m' = B_1 ... B_l$. Note that if A_i contains only 1, then s must have the same number of 1's in the i-th block, otherwise there will be some $s[k]$ for which $\deg(s[k]) > 1$, which contradicts Claim D.2. Similarly, if B_j contains only 0, then s has the same number of 0's in the j-th block. Then we can fully recover s as $h(A_1) ... h(A_l)$ where $h(X_i) = A_i$ if X_i contains 1 and $h(X_i) = B_i$ if X_i contains 0. ■

Zhuangfei Hu is a M.Math student in the David R. Cheriton School of Computer Science at University of Waterloo. Prior to that he received B.S. in Computer Science and Technology in Tsinghua University in 2020. His research interests include distance algorithms, optimization and quantum computing.

Xinda Li is currently a research associate at the David R. Cheriton School of Computer Science at the University of Waterloo. He received his M.Math and B.S. degree in Computer Science from the University of Waterloo in 2022 and 2020. His research interests include data security, privacy and machine learning.

David P. Woodruff is a professor at Carnegie Mellon University in the Computer Science Department. Before that he was a research scientist at IBM Almaden for ten years. He received his PhD from MIT in 2007. His research interests include data stream algorithms, distributed algorithms, machine learning, numerical linear algebra, optimization, sketching, and sparse recovery. He is the recipient of the 2020 Simons Investigator Award, the 2014 Presburger Award, Best Paper Awards at STOC 2013, PODS 2010, and PODS, 2020, and a STOC 2023 Test of Time Award. At IBM he was a member of the Academy of Technology and a Master Inventor.

Hongyang Zhang (Member, IEEE) is an assistant professor at University of Waterloo in the David R. Cheriton School of Computer Science. He received his PhD from Carnegie Mellon University in 2019. After that, he was a postdoc research associate at Toyota Technological Institute at Chicago. His research interests include machine learning and AI security.

Shufan Zhang (Student Member, IEEE) received the M.Math degree from the University of Waterloo, Waterloo, ON, Canada, in 2022. He is currently working toward the Ph.D. degree in computer science at the University of Waterloo. His research interests include computer security and data privacy, on both theory and system aspects, as well as their intersections with database systems and machine learning.