Privacy-Preserving Search of Similar Patients in Genomic Data

Gilad Asharov*	Shai Halevi†	Yehuda Lindell‡	Tal Rabin†
Cornell Tech	IBM Research	Bar-Ilan University	IBM Research
asharov@cornell.edu	shaih@alum.mit.edu	Yehuda.Lindell@biu.ac.il	talr@us.ibm.com

June 1, 2017

Abstract

The growing availability of genomic data holds great promise for advancing medicine and research, but unlocking its full potential requires adequate methods for protecting the privacy of individuals whose genome data we use. One example of this tension is running Similar Patient Query on remote genomic data: In this setting a doctor that holds the genome of his/her patient may try to find other individuals with "close" genomic data, and use the data of these individuals to help diagnose and find effective treatment for that patient's conditions. This is clearly a desirable mode of operation, however, the privacy exposure implications are considerable, so we would like to carry out the above "closeness" computation in a privacy preserving manner.

Secure-computation techniques offer a way out of this dilemma. However, applying state-of-the-art MPC to the most optimized existing edit distance algorithm comes at a high cost and is not practical. Wang et al. proposed recently (ACM CCS '15) an efficient solution, for situations where the genome sequences are so close that edit distance between two genomes can be well approximated just by looking at the indexes in which they differ from the reference genome. However, this solution does not extend well to cases with high divergence among individual genomes, and different techniques are needed there.

In this work we put forward a new approach for highly efficient secure computation for computing an approximation of the edit-distance, that works well even in settings with much higher divergence. We present contributions on two fronts. First, the design of an approximation method that would yield an efficient private computation. Second, further optimizations of the two-party protocol. Our tests indicate that the approximation method works well even in regions of the genome where the distance between individuals is 5% or more with many insertions and deletions (compared to 99.5% similarly with mostly substitutions, as considered by Wang et al.). As for speed, our protocol implementation takes just a few seconds to run on databases with thousands of records, each of length thousands of alleles, and it scales almost linearly with both the database size and the length of the sequences in it. As an example, in the datasets of the recent iDASH competition, it takes less than two seconds to find the nearest five records to a query, in a size-500 dataset of length-3500 sequences. This is 2-3 orders of magnitude faster than using state-of-the-art secure protocols with existing edit distance algorithms.

Keyword: Genomic privacy, cryptographic protocols, edit-distance

*Some of the work was done while the author was a post-doctoral researcher at IBM T.J. Watson Research Center. Previously supported by NSF Grant No. 1017660. Currently supported by a Junior Fellow award from the Simons Foundation.

†Supported in part by the Defense Advanced Research Projects Agency (DARPA) and Army Research Office(ARO) under Contract No. W911NF-15-C-0236.

‡Supported by the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

1 Introduction

Consider the task of a medical doctor who wants to compare a patient's DNA against a remote genomic database, e.g., to determine the patient's pre-disposition to various medical conditions. The database contains a list of individual genome sequences, each labeled with the medical conditions of that person. The doctor needs to find the few individuals in the database whose genome sequence (in the relevant segment) most resembles that of the patient, and learn the medical conditions of these individuals. We define resemblance (or closeness) in terms of edit distance. Sending the patient's DNA sequence to the database has severe privacy implication, thus, we would like to find an effective privacy preserving solution to this task.

More specifically, we seek a solution to the following k-closest-match problem: We have a server holding a database DB of genomic sequences $(S_1, \ldots, S_m)$, whose approximate length and position inside the human genome are known. The client (doctor) is holding a sequence query Q, and wishes to find the identities of the k sequences in the DB that have the smallest edit distance from Q (where k is a public parameter). The goal is to perform this computation in a privacy-preserving manner. We target security in the presence of an honest-but-curious adversary. Our work was motivated by the recent "secure genome analysis competition" run by iDASH [iDA16] (in which we won the 1st place for accuracy and speed).

Unfortunately, the straightforward solution of computing the exact edit distancei of Wagner-Fischer [WF74] using a secure-computation protocol (or even the near-linear-time approximation of Andoni and Onak [AO12]) will be prohibitively slow. Using state-of-the-art secure-computation techniques, such protocols would take many minutes (maybe even hours) per query, and certainly will not scale to large datasets and long sequences.

In this work we develop an efficient privacy-preserving protocol for computing an approximation of the k-match function from above. Our solution lets the client and server preprocess their respective inputs to the protocol, in a manner that makes the secure computation portion of the work much less expensive. While the preprocessing can include many edit-distance computations, these are all carried out in the clear, and so they are much cheaper than the secure computation portions. Moreover, this preprocessing is reusable and can be used by the server to answer unlimited number of queries (and similarly by the client to query multiple databases).

We show that the implementation of this approximation can handle databases with hundreds of records and sequences of length thousands of alleles. We ran our solution on a few databases of various sizes in regions featuring high divergence among individual genomes (variability of around 5%). Our experiments yielded excellent results both with respect to the accuracy and runtime (see Section 4 for details). Furthermore, our protocol was tested by external referees as part of the participation in the iDash competition, in which we won the first place for the fastest runtime and for accuracy. Similar accuracy and performance results to the values that we report were confirmed. After a one-time preprocessing of around 12 seconds, our solution can answer many queries in less than a second each.

The databases that are accessible for real genomic data include in the range of hundreds of genomes. We wanted to "stress-test" our implementation using a larger dataset, consisting of 10,000 sequences. For this we received a synthesized database which had more variability than the real ones (variability under 10%). As this was not real genomic material this presented some complications, we detail all of these in Section 4.3.

We note that the previous solutions for this problem that focus on regions with much less divergence (overall variability of 0.5%) are not sensitive enough to approximate well the distances in regions with higher diversity as addressed in this paper.

1.1 Overview of Our Solution

Our goal is to design a protocol whose two-party computation portion is as small as possible as this is the computation intensive part. For this we create a new (approximate) edit distance protocol that requires the holder of the database to carry out many full edit distance computations and the query holder to compute a single edit distance computation. However, all these computations are done in the clear by each one of the parties locally and thus the runtime of these edit distances is negligible yielding a highly efficient privacy preserving secure computation for the k-closest match problem.

1.1.1 Approximation Function

We develop an efficient approximation algorithm that utilizes the distribution of genomic data. We heuristically expect (and empirically verify) that the our algorithm provides an excellent approximation of the desired functionality.

We first replace the edit-distance function with a block-wise approximation of it. Using a specially tailored method (described below) we break the query Q into n blocks $(Q_1, \ldots, Q_n)$, and similarly break each sequence S_i in the database into blocks $(S_{i,1}, \ldots, S_{i,n})$, where the blocks are very small (typically, no more than 15 letters). Denoting by ED the edit-distance function, we roughly use the approximation

$$\mathsf{ApproxED}(Q, S_i) \approx \sum_{\ell=1}^{n} \mathsf{ED}(Q_\ell, S_{i,\ell}). \tag{1.1}$$

This approximation alone reduces the cost significantly, as computing the distance of $\mathsf{ED}(Q_\ell, S_{i,\ell})$ when $|Q_\ell|, |S_{i,\ell}|$ are very small is cheap, and so the computation of $\mathsf{ApproxED}(Q, S_i)$ can be done in linear time. However, we can still do much better: We observe that for our genomic data, each block position has only a few distinct values (such as $\{\mathtt{TT}, \mathtt{AGT}, \mathtt{AGG}\}$) that actually appear there. Namely, for each $\ell : \ell = 1, ..., n$ the set of values $T_\ell = \{S_{i,\ell} : i = 1, \ldots, m\}$ is much smaller than m. In our tests, even for our largest database with $m = 10,000$, we only had $v = \max_\ell |T_\ell| \leq 40$. This means that the $\mathsf{ED}(Q_\ell, S_{i,\ell})$ needs to be computed only for 40 values rather than $10,000$ (saving 99.5% amount of the work). Furthermore, in almost all cases ($> 99\%$), the block Q_ℓ of the query is also one of the values in the set T_ℓ implying that the edit distance values are from the set $\mathsf{ED}(u, S_{i,\ell})$ for all $u \in T_\ell$.

Let v be some known bound on the number of distinct values in each block (e.g., $v = 40$). We define a bit variable $\chi_{\ell,j}$ that indicates whether or not $u_{\ell,j} = Q_\ell$, for $u_{\ell,j} \in T_\ell$. If the value Q_ℓ happens to be equal to one of the $u_{\ell,j}$'s, then for every $S_{i,\ell}$ we have

$$\mathsf{ED}(Q_\ell, S_{i,\ell}) = \sum_{j=1}^{v} \chi_{\ell,j} \cdot \mathsf{ED}(u_{\ell,j}, S_{i,\ell}). \tag{1.2}$$

Namely, in this case we can compute the values that are needed for Eq. (1.1) as a simple linear combination involving the (few) bits $\chi_{\ell,j}$ and the values $\mathsf{ED}(u_{\ell,j}, S_{i,\ell})$. Thus, we approximate the edit distance between Q and S_i using

$$\mathsf{ApproxED}(Q, S_i) = \sum_{\ell=1}^{n} \sum_{j=1}^{v} \chi_{\ell,j} \cdot \mathsf{ED}(u_{\ell,j}, S_{i,\ell}). \tag{1.3}$$

We note that in the case where $Q_\ell \notin T_\ell$, the expression on the right-hand side of Eq. (1.2) is always zero, so these cases introduce more error to our approximation. However, we verified

empirically that the effect of this added error is very minor (see Appendix B.2). Summing up, we compute securely the approximate k-closest-match function, defined as:

$$\begin{aligned} &\mathsf{ApproxClosest}_{k,m}(Q, \{S_1, \ldots, S_m\}) \\ &\quad = \text{ indexes } i_1, \ldots, i_k \text{ of the } k \text{ } S_i\text{'s with min } \mathsf{ApproxED}(Q, S_i), \end{aligned} \tag{1.4}$$

where ties are broken using the indexes i themselves.

This function will lend itself very naturally to an efficient MPC protocol as many of the elements in this function can be computed locally in the clear by the owner of the dataset (see below).

1.1.2 Partitioning into Blocks

A crucial detail of our approximation is the method that we use to partition the sequences S_i and the query Q into blocks. The simplest possibility would be to partition them into fixed-length blocks, but this simplistic partitioning yields a poor approximation. For example, a small shift close to the beginning of the query (perhaps just inserting a single character) can lead to many misalignments in the consecutive blocks, causing this single error to be counted multiple times.

To get a better partitioning method, we utilize specific features of our application domain,[1] specifically the existence of a public "reference genome" R, which is somewhat close to both the query Q and the database sequences S_i. This genome is known as the Standard Reference Assembly GRCh37 [Wik17]. Our partitioning method begins with applying the simplistic partitioning above to the reference genome R. Then, each party separately aligns its input sequence(s) to the reference sequence (using the Wagner-Fischer algorithm [WF74]), and we hope that the alignments of Q vs. R and S_i vs. R are "close enough" to induce a good alignment between Q and S_i.

Using this publicly known sequence for partitioning yields very good results on the databases which we generated and on the data of the iDASH competition (that has variability under 5%): In our tests, our approximation algorithm returned exactly the k closest sequences in 98% of the runs, and a very good approximation in the remaining 2%, see more details in Section 4. Our experiments show similar results for other regions of the genome.

We stress that the alignment of Q and the S_i's to R is done locally *and in the clear* by each party. Description of this procedure and an estimate of its accuracy can be found in Section 3.

1.1.3 Efficient Secure Computation

Transforming the approximation procedure above into a secure protocol is not a straightforward application of generic transformations (e.g., Yao [Yao86] or GMW [GMW87]). Rather we use the specific form of our approximation to get a significantly faster implementation.

The protocol begins with a preprocessing phase. Examining Eq. (1.3) we observe that the holder of the dateset can carry out much of the computation locally in the clear. The server first breaks all the genomes into blocks as described above, and then it computes the sets T_ℓ and all the intra-block edit-distance values defining a matrices L_ℓ such that $L_\ell[j,i] = \mathsf{ED}(u_{\ell,j}, S_{i,\ell})$, for $\ell = 1, ..., n, j = 1, .., v, i = 1, ..., m$.

Once this matrices are computed and held by the server in the clear this transforms the computation of Eq. (1.3) into a matrix-vector multiplication. That is, the above matrices and the vector of bits $\chi_{\ell,j}$ should be multiplied. As the values of these bits, $\chi_{\ell,j}$, need to remain secret we need to compute these bits using a secure protocol.

[1] We remark that using application-specific partitioning method is the best we can hope for: Any general-purpose partitioning that yields linear-time processing (and guarantee accuracy) will violate a conditional lower bound on the complexity of edit-distance calculations [BI15].

To compute the vector of shared bits $\chi_{\ell,j}$ the parties engage in a standard secure computation protocol for computing a random XOR sharing of all the bits $\chi_{\ell,j}$, using an optimized variant of Yao's garbled circuits. Then for each j, ℓ the parties execute a 1-out-of-2 oblivious transfer protocol to get a random additive sharing of the value $\chi_{\ell,j} \cdot L_\ell[j,i]$, which is of course accelerated using OT-extension [ALSZ13, KOS15].

The parties then locally sum up their shares as per Eq. (1.3), thus obtaining an additive sharing of the m approximate edit distance values $\mathsf{ApproxED}(Q, S_i)$. Finally a standard secure computation protocol, using an optimized variant of Yao's garbled circuits, yields the indexes of the k smallest values. We prove that

Theorem 1.1 (informal)**.** *The protocol sketched above securely computes the function* $\mathsf{ApproxClosest}_{k,m}$ *from Eq. (1.4) in the semi-honest adversary model.*

1.2 Implementation and Performance

We implemented our protocol using the `C++` version of the Secure Computation API library (SCAPI) [EFLL12], and tested it on a few databases with hundreds of real genomic sequences. Furthermore, the protocol was evaluated by external referees as part of the iDASH competition. To see whether the protocol could withstand the stress of a much larger database we ran it on a synthesized dataset with thousands of records (details in Section 4.3).

In our tests, the most costly aspect was the pre-processing on the server side (which is preformed in the clear, and only needed to be done once). We did not optimize this part and it took under 12 seconds for our 500-sequence database (with the length of each sequence $w \approx 3500$). We expect an optimized implementation to be much faster (perhaps by an order of magnitude).

For the online secure computation itself (which is done for every query), the overall number of non-XOR gates is only about 800K AND gates, and we use roughly the same number of OTs. Using efficient implementations of Yao's garbled circuits [KS08, ZRE15] and OT extensions [ALSZ13, KOS15], it took about 1 seconds to process each query.

1.3 Related Work

Jha et al. proposed in [JKS08] some techniques for secure edit distance using garbled circuits, and shows that the overhead is acceptable only for small strings. (For example, handling 200-character strings takes about 2GB of bandwidth). Using some further optimizations, they showed that 500-character string instances can be computed in almost an hour. Computing edit distance is also a common benchmark for analyzing improvements in general secure computation techniques and frameworks (see [HEKM11, HSE+11, ALSZ13], to state a few). These works compute accurate edit distance, and do not utilize the specific input distribution of genomic data.

Recent years saw a large body of work on using secure computation protocols for genomic data, some surveys include [NAC+15, ABOcS15]. The most relevant prior work to ours is by Wang et al. [WHZ+15] (building on earlier work of Baldi et al. [BBC+11]), that considers the same task: designing a privacy-preserving protocol for supporting the Similar Patient Query. Wang et al. developed an approximation protocol for edit distance that enables computation in a large scale of the *whole* genome (i.e., supporting up to 80M nucleotide, all locations in the genome that are suspected to have variations). The approximation is shown to be very accurate, and the computation is performed in several minutes.

The approach of Wang et al. relies heavily on the fact that the genomes that they are examining have little divergence, i.e. not more than 0.5% distance between genomes and that most differences (80-90%) are substitutions. Using this fact, Wang et al. show how to approximate the edit distance

by just considering the set of indexes where the two sequences differ from the reference genome, and running a set-intersection protocol.

The above assumptions are valid in some instances, however, some regions have high divergence and the differences are also caused by insertions and deletions, which are more difficult to deal with in computing edit distance. For example in some regions that affect the immune system the distances between two individuals may be up to 5–7% (of the size of the region), and about 25% of the differences between two sequences are due to insertions or deletions. Hence the approach of Wang et al. cannot be used and a different method is needed. We remark that most regions have much lower variability.

In this work we provide a solution that works also for regions with high-divergence and high insertion/deletion rates (as needed for our motivating application of identifying likely conditions based on a small portion of the human genome). We note that the two methods can be combined in applications: Large regions with small variability can be approximated using the more coarse method of Wang et al., while other (smaller) regions with higher variability can be approximated using our more sensitive method.

Security implications of computing an approximation. Feigenbaum et al. observed [FIM+01] that computing an approximated version of a function may have security implications, in that the approximated version may leak information which is not revealed by the exact version. This concern certainly applies to our solution. For example it is not hard to see that by engaging in multiple executions with adversarially chosen queries, a client can fully recover the sets of block values T_ℓ from above.

It is interesting to ask to what extent this leakage is problematic in real life applications (and how it can be mitigated), but such questions go beyond the scope of the current work.

Organization. The rest of this paper is organized as follows. We reverse the order of the sections relative to the order in the Introduction and start with the presentation of the secure computation in Section 2. Followed by the description of how to break the sequences into blocks in Section 3, and describe also an algorithm for generating the reference sequence. We report the accuracy of our protocol in Section 4. We conclude with the implementation in Section 5. In the appendixes we report provide some of our experiments, as well as some supplementary data for decision we made in our design.

2 Privacy Preserving Protocol

In this section we present our semi-honest secure protocol for computing the ApproxClosest function from Eq. (1.4): The client has a query string, the server has a database of records, and the client needs to learn the indices (or labels) of the k closest records to its query, as specified in Functionality 2.1 below.

As described in the Introduction we do not compute the exact edit distance between the query and the sequences in the database, but rather an approximation of this value which is amenable to an efficient secure computation. The exact function that we compute depends on our procedure for breaking the sequences and query into blocks, which we describe in detail in Section 3 below. That procedure computes the blocks $Q = (Q_1, \ldots, Q_n)$ and $S_i = (S_{i,1}, \ldots, S_{i,n})$ (where n is a known parameter, in out implementation typically $n = 1200$). For each block location we define a set, T_ℓ, of values that occur in that block position, that is

$$T_\ell = \{S_{1,\ell}, \ldots, S_{m,\ell}\}.$$

Functionality 2.1: (Approximate) Closest k Records Functionality

- **Public parameters:** The database size m and output size $k < m$.
- **Private inputs:** The client holds a sequence query Q. The server holds a database DB of m sequences $(S_1, \ldots, S_m)$.
- **The functionality:**
  1. Let $\tilde{e}_i = \mathsf{ApproxED}(Q, S_i)$ be the approximate edit distance between Q and S_i, as per Eq. (2.1).
  2. Let I_k be the set of indexes of the k-smallest values in $\tilde{e}_1, \ldots, \tilde{e}_m$, breaking ties according to the lexicographic order.
- **Output:** The client outputs I_k (ordered lexicographically), the server has no output.

The approximate edit distance function that we compute is:

$$\begin{aligned} \mathsf{ApproxED}(Q, S_i) &= \sum_{\ell=1}^{n} \Delta(Q_\ell, S_{i,\ell}), \text{ where} \\ \Delta(Q_\ell, S_{i,\ell}) &= \begin{cases} \mathsf{ED}(Q_\ell, S_{i,\ell}) & \text{If } Q_\ell \in T_\ell \\ 0 & \text{otherwise .} \end{cases} \end{aligned} \tag{2.1}$$

We remark that although computing $\mathsf{ED}(Q_\ell, S_{i,\ell})$ also for blocks where $Q_\ell \notin T_\ell$ would improve accuracy, this improvement is minor (since the case of $Q_\ell \notin T_\ell$ is rare). See Appendix B.2 for further discussion of our choice. Our protocol for realizing Functionality 2.1 consists of a local preprocessing stage, followed by two main protocol stages:

Preprocessing: In this stage the parties break their sequences in to blocks, and the server computes several tables. We describe this stage in Section 2.1.

Stage I: computing additive sharing of the approximations. This stage is the crux of our protocol, and is described in Section 2.2. The client and the server interactively compute secret sharing of the vector of approximated edit distances. Specifically, the parties compute additive sharing of the following vector L:

$$L \stackrel{\Delta}{=} (\mathsf{ApproxED}(Q, S_1), \ldots, \mathsf{ApproxED}(Q, S_m)). \tag{2.2}$$

Stage II: computing the k-minimal values. In the second stage of the interaction, the client and the server compute the k minimal values of the secret-shared vector L, and learn the indices of these values. This stage is described in Section 2.3.

2.1 The Preprocessing Stage

The preprocessing stage relies on a procedure $\mathsf{BreakToBlocks}$ that the two parties use to break each of their respective sequences into blocks. That procedure is described in Section 3, and it has the property that for each block location there are only a few distinct values that occur there, and moreover that the two parties know a bound v on the number of values in each block. The $\mathsf{BreakToBlocks}$ procedure is parametrized by public reference sequence R and blocksize parameter b.

The client. On input the query Q, the client sets $(Q_1, ..., Q_n) := \mathsf{BreakToBlocks}_{R,\mathsf{b}}(Q)$.

The server. On input the database, $S_1, ..., S_m$, the server proceeds as follows:

1. Set $(S_{i,1}, ..., S_{i,n}) := \mathsf{BreakToBlocks}_{R,\mathsf{b}}(S_i)$ for all $i = 1, \ldots, m$.

2. For each block location $\ell = 1, ..., n$, compute the set
$$T_\ell = \{S_{i,\ell} : i = 1, \ldots, m\} = \{u_{\ell,1}, \ldots, u_{\ell,v}\} \tag{2.3}$$
of all the values in the ℓth block. The server pads all sets T_ℓ to be of the same size v using some dummy values.

3. For every block location $\ell = 1, \ldots, n$, every sequence S_i, $i = 1, \ldots, n$, and every value $u_{\ell,j} \in T_\ell$, $j = 1, \ldots, v$, the server computes the edit distance between $u_{\ell,j}$ and $S_{i,\ell}$, setting $L_\ell[j, i] := \mathsf{ED}(u_{\ell,j}, S_{i,\ell})$. Below we denote the row $L_\ell[j, \cdot]$ by $L_{\ell,j}$, namely
$$L_{\ell,j} := (\mathsf{ED}(u_{\ell,j}, S_{1,\ell}), \ldots, (\mathsf{ED}(u_{\ell,j}, S_{m,\ell}))\,. \tag{2.4}$$
(Jumping ahead, each vector $L_{\ell,j}$ represents the contribution of the ℓ'th block to the final edit distances approximations, for the case where $Q_\ell = u_{\ell,j}$.)

The preprocessing of the server is done only once, and then multiple queries can be computed.

Computing Eq. (2.1). We observe that for each i, ℓ, the value $\Delta(Q_\ell, S_{i,\ell})$ from Eq. (2.1) can be expressed as
$$\Delta(Q_\ell, S_{i,\ell}) = \sum_{j=1}^{v} \chi_{\ell,j} \cdot \underbrace{\mathsf{ED}(u_{\ell,j}, S_{i,\ell})}_{=L\ell[j,i]}, \tag{2.5}$$
where,
$$\chi_{\ell,j} \triangleq \begin{cases} 1 & \text{if } Q_\ell = u_{\ell,j} \\ 0 & \text{otherwise} \end{cases}.$$
Therefore we have for all i
$$\mathsf{ApproxED}(Q, S_i) = \sum_{\ell=1}^{n} \Delta(Q_\ell, S_{i,\ell}) = \sum_{\ell=1}^{n} \sum_{j=1}^{v} \chi_{\ell,j} \cdot L_\ell[j, i].$$
Thus, the vector of approximations $(\mathsf{ApproxED}(Q, S_i))_i$ can be computed as
$$L = \big(\mathsf{ApproxED}(Q, S_1), \ldots, \mathsf{ApproxED}(Q, S_m)\big) \;=\; \sum_{\ell=1}^{n} \sum_{j=1}^{v} \chi_{\ell,j} \cdot L_{\ell,j}\ . \tag{2.6}$$

2.2 Stage I: Computing Additive Sharing of the Vector L

After preprocessing, the client holds a vector of blocks $(Q_1, ..., Q_n)$, and the server holds all the (ordered) sets $T_1, \ldots, T_n$ and the edit-distance vectors $L_{\ell,j}$ for $\ell = 1, \ldots, n$ and $j = 1, \ldots, v$. Our goal in the first stage of interaction is to compute additive sharing of the approximate-distance vector L. Formally, we need to realize the Functionality 2.2 below:
The protocol for realizing Functionality 2.2 consists of two main steps:

- First, the parties compute shares of the indicator bits $\chi_{\ell,j}$. That is, for every $\ell \in [n], j \in [v]$, the client and server receive random bits $\chi^c_{\ell,j}$, $\chi^s_{\ell,j}$, respectively, s.t. $\chi^c_{\ell,j} \oplus \chi^s_{\ell,j} = \chi_{\ell,j}$.
- Next they use oblivious transfer to convert their shares of $\chi_{\ell,j}$ (and the value $L_{\ell,j}$ held by the server) into additive shares of $\chi_{\ell,j} \cdot L_{\ell,j}$. That is, they choose random vectors $L^c_{\ell,j}, L^s_{\ell,j}$ s.t. $L^c_{\ell,j} - L^s_{\ell,j} = \chi_{\ell,j} \cdot L_{\ell,j} \pmod d$.

Then the client and server locally sum their shares: The client is computing $L^c = \sum_{\ell=1}^{m} \sum_{j=1}^{v} L^c_{\ell,j} \bmod d$, and the server $L^s = \sum_{\ell=1}^{m} \sum_{j=1}^{v} L^s_{\ell,j} \bmod d$. Hence
$$L^c - L^s = \sum_{\ell=1}^{n} \sum_{j=1}^{v} L^c_{\ell,j} - L^s_{\ell,j} = \sum_{\ell=1}^{n} \sum_{j=1}^{v} \chi_{\ell,j} \cdot L_{\ell,j} = L \pmod d.$$

Functionality 2.2: Additive Sharing of Approximate Edit-Distances, $L^c - L^s = L$

- **Parameters**: Let d be a public upper bound on $\max_{i \in m} \mathsf{ApproxED}(Q, S_i)$.
- **Input:** The client inputs the blocks $(Q_1, \ldots, Q_n)$. The server inputs the tables $T_\ell = \{u_{\ell,1}, \ldots, u_{\ell,v}\}_{\ell \in [n]}$ and vectors $\{L_{\ell,j}\}_{\ell \in [n], j \in [v]}$.
- **The functionality:**
  1. Let $L = \sum_{\ell=1}^{n} \sum_{j=1}^{v} \chi_{\ell,j} \cdot L_{\ell,j} \in d^m$
     ($L_{\ell,j}$, $\chi_{\ell,j}$ are defined in Eq. (2.4), Eq. (2.5), respectively);
  2. Choose a random vector $L^s \in d^m$ and set $L^c := L + L^s \bmod d$.
- **Output:** The client outputs L^c while the server outputs L^s.

2.2.1 Step 1: Indicator Bits

This step realizes the Functionality 2.3:

Functionality 2.3:
Computing XOR sharing for the indicator bit ($\chi^c_{\ell,j} \oplus \chi^s_{\ell,j} = \chi_{\ell,j}$)

- **Input:** The client inputs the block Q_ℓ.
  The server inputs $u_{\ell,j}$, which is the jth value in the set T_ℓ.
- **The functionality:** Let $\chi_{\ell,j} = 1$ if $Q_\ell = u_{\ell,j}$, and $\chi_{\ell,j} = 0$ otherwise.
  Choose a random bit $\chi^s_{\ell,j}$ and set $\chi^c_{\ell,j} = \chi^s_{\ell,j} \oplus \chi_{\ell,j}$.
- **Output:** The client outputs $\chi^c_{\ell,j}$ while the server outputs $\chi^s_{\ell,j}$.

We realize Functionality 2.3 using a direct application of Yao's protocol. Let $Q_\ell = \sigma_1, \ldots, \sigma_t$ and $u_{\ell,j} = \tau_1, \ldots, \tau_t$, representing the inputs $Q_\ell, u_{\ell,j}$ (each padded to some bound b' and converted to binary using suffix-free encoding).[2] The server chooses a random bit $\chi^s_{\ell,j}$ (which will also be its output of the protocol), and we use a standard secure protocol (e.g., Yao's protocol) in which the client learns the output bit $\chi^c_{\ell,j} = \chi^s_{\ell,j} \oplus \bigwedge_{k=1}^{t} (\sigma_k \oplus \tau_k \oplus 1)$. At the end of this stage, the client and the server holding bits $\chi^c_{\ell,j}, \chi^s_{\ell,j}$ (resp.) for every $\ell = 1, \ldots, m$ and $j = 1, \ldots, v$.

2.2.2 Step 2: Additive sharing

This step realizes the Functionality 2.4.

Functionality 2.4: Computing additive sharing for $\chi_{\ell,j} \cdot L_{\ell,j}$

- **Parameters**: The edit-distance bound d.
- **Input:** Client inputs is $\chi^c_{\ell,j}$, server inputs is $\chi^s_{\ell,j}$ and the vector $L_{\ell,j}$.
- **The functionality:** Set $\chi_{\ell,j} = \chi^c_{\ell,j} \oplus \chi^s_{\ell,j}$. Choose a random vector $L^s_{\ell,j} \in d^m$ and set $L^c_{\ell,j} = L^s_{\ell,j} + \chi_{\ell,j} \cdot L_{\ell,j}$.
- **Output:** The client outputs $L^c_{\ell,j}$ and the server outputs $L^s_{\ell,j}$.

We realize Functionality 2.4 using oblivious transfer, as described in Protocol 2.5.

2.2.3 Putting it together: realizing Functionality 2.2

We realize functionality 2.2 in Protocol 2.6 using functionalities 2.3 and 2.4.

[2] In our case the original strings were over a 4-ary alphabet, so to get suffix-free encoding we need to set at least $t = 2\mathsf{b}' + 1$.

Protocol 2.5: Realizing Functionality 2.4 (in the OT-hybrid model)

- **Parameters**: The edit-distance bound d.
- **Input:** Client inputs is $\chi^c_{\ell,j}$, server inputs is $\chi^s_{\ell,j}$ and the vector $L_{\ell,j}$.
- **The protocol:** (all additions are done (mod d))
  1. The server chooses a random vector $L^0_{\ell,j}$ and sets $L^1_{\ell,j} = L^0_{\ell,j} + L_{\ell,j}$.
  2. The server and the client engage in a 1-out-of-2 oblivious transfer. The client as the receiver with the choice bit $\chi^c_{\ell,j}$, and the server as the sender with inputs:
     - $(L^0_{\ell,j}, L^1_{\ell,j}) = (L^0_{\ell,j}, L^0_{\ell,j}) + (0, L_{\ell,j})$ in case $\chi^s_{\ell,j} = 0$,
     - $(L^1_{\ell,j}, L^0_{\ell,j}) = (L^0_{\ell,j}, L^0_{\ell,j}) + (L_{\ell,j}, 0)$ in case $\chi^s_{\ell,j} = 1$.

     Let $L^c_{\ell,j}$ denote the output that the client receives from the OT protocol.
- **Output:** The server outputs $L^s_{\ell,j} = L^0_{\ell,j}$ and the client outputs $L^c_{\ell,j}$.

Protocol 2.6: Realizing Functionality 2.2

- **Parameters**: Let d be a public upper bound on $\max_{i \in m} \mathsf{ApproxED}(Q, S_i)$.
- **Input:** The client inputs the blocks $(Q_1, \ldots, Q_n)$. The server inputs the tables $T_\ell = \{u_{\ell,1}, \ldots, u_{\ell,v}\}_{\ell \in [n]}$ and vectors $\{L_{\ell,j}\}_{\ell \in [n], j \in [v]}$.
- **The protocol:** (all additions are done mod d)
  1. For every $\ell = 1, \ldots, n$ and $j = 1, \ldots, v$:
     (a) Invoke Functionality 2.3, with client input Q_ℓ and server input $u_{\ell,j}$. Let $\chi^c_{\ell,j}$, $\chi^s_{\ell,j}$ be the outputs of the client and server, respectively.
     (b) Invoke Functionality 2.4 with client input $\chi^c_{\ell,j}$ and server input the bit $\chi^s_{\ell,j}$ and the vector $L_{\ell,j}$. Let $L^c_{\ell,j}, L^s_{\ell,j}$ be the output of the client and server, respectively.
  2. The client computes $L^c = \sum_{\ell=1}^{n} \sum_{j=1}^{v} L^c_{\ell,j}$,
     the server computes $L^s = \sum_{\ell=1}^{n} \sum_{j=1}^{v} L^s_{\ell,j}$.
- **Output:** The client outputs L^c, the server outputs L^s.

2.3 Stage II: Finding k Minimal Values

After computing an additive sharing of the approximate edit distances between Q and the m records $S_1, \ldots, S_m$, the parties engage in a protocol to find the k smallest distances. The full specification is found in Functionality 2.7.

Functionality 2.7: Find the k-Minimal Values

- **Parameters:** number of records m, output size k, distance bound d.
- **Input:** Client and server hold $L^c, L^s \in \mathbb{Z}_d^m$, respectively.
- **The functionality:**
  1. Let $L = L^c - L^s \bmod d$, and denote $L = (L_1, \ldots, L_m)$
  2. Find the k smallest values in the sequence L, using the indexes $1, \ldots, m$ to break ties.

  Output: The client gets m bits $(\sigma_1, \ldots, \sigma_m)$, where $\sigma_j = 1$ if $L[j]$ is one of the k smallest values. The server has no output.

The protocol to realize Functionality 2.7 is just a direct application of Yao's protocol, applied to the simple circuit that computes $L = (L_1, \ldots, L_m) = L^c - L^s \bmod d$, then repeatedly finds the minimum k times (breaking ties using the index).[3]

[3]Breaking ties according to index is easy when using a comparison tree with the leaves ordered by their index, you just always break ties in favor of the left child.

Realizing Functionality 2.1 is now a straightforward application of the components above, as summarized in Protocol 2.8 below.

Protocol 2.8: Realizing Functionality 2.1

- **Parameters:** Database size m, output size $k < m$, distance bound d.
- **Input:** The client holds a sequence query Q. The server holds a database DB of m sequences $(S_1, \ldots, S_m)$.
- **The protocol:**
  1. The clients and the server perform the preprocessing stage. The client holds the blocks $Q_1, \ldots, Q_n$, and the server holds the tables $T_1, \ldots, T_\ell$ and the vector $\{L_{\ell,j}\}_{\ell \in [n], j \in [v]}$.
  2. The parties invoke Functionality 2.2, where the client inputs $Q_1, \ldots, Q_n$ and the server inputs the vector $\{L_{\ell,j}\}_{\ell \in [n], j \in [v]}$ and the tables $\{T_\ell\}_{\ell=1}^{n}$.
     The client receives vector L^c and the server receives the vector L^s.
  3. The parties invoke Functionality 2.7, where the client inputs L^c and the server inputs L^s. The client receives the m bits $(\sigma_1, \ldots, \sigma_m)$.
- **Output:** The client outputs $(\sigma_1, \ldots, \sigma_m)$.

2.4 Security Analysis

Below we sketch the security analysis of our protocol. We follow the standard definition of static semi-honest security in the standalone model (cf. [Gol04]). Namely we need to describe a simulator for the two cases of corrupted client and corrupted server, that gets the input and output of the corrupted party and needs to generate a protocol view that is indistinguishable from the view of the real protocol. For randomized functionality we consider the joint distribution of the view of the corrupted party and the output of all parties. We argue the security in a bottom-up fashion:

- We assume semi-honest security of the basic building blocks that we use, i.e., oblivious-transfer and Yao's protocol.
- This immediately implies the security for the find-k-min protocol for realizing Functionality 2.7, as well as the security of the protocol for realizing Functionality 2.3 (as both of these are direct applications of Yao's protocol).
- Next we prove the security of Protocol 2.5 that realizes Functionality 2.4 (sharing of $\chi_{\ell,j} \cdot L_{\ell,j}$), and then Protocol 2.6 that realizes Functionality 2.2 (sharing of L).
- Finally we put everything together and prove security of the entire protocol (Protocol 2.8).

Note that only the last two bullets require new proofs, everything else holds by assumption on the components that we use.

Theorem 2.9 (Sharing of $\chi_{\ell,j} \cdot L_{\ell,j}$)**.** *Assuming the semi-honest security of the underlying* 1*-out-of-*2 *OT protocol, the Protocol 2.5 securely realizes Functionality 2.4 against static corruptions in the semi-honest adversary model.*

Proof Sketch: Correctness if easy to see, just by inspecting the output in the four cases $(\chi^c_{\ell,j}, \chi^s_{\ell,j}) \in \{(0,0),(0,1),(1,0),(1,1)\}$.

Regarding privacy, as the protocol is just an execution of an OT protocol, the view of the server is its random tape, $L^s_{\ell,j}$. Thus, in order to simulate a corrupted server, the simulator, upon receiving the output $L^s_{\ell,j}$ from the functionality, just outputs this vector. In case of a corrupted client, the view of the client is just its final output. The theorem follows. ∎

Theorem 2.10 (Sharing of L)**.** *Protocol 2.6 securely realizes Functionality 2.2 against static corruptions in the semi-honest adversary model.*

Proof: Correctness is easy by inspection. As for security, assume the case of a corrupted client. The simulator receives a random L^c as the output of the corrupted client, and the honest server receives $L^s = L^c - L$. The view of the client during the execution is the set of shares $\{\chi^c_{\ell,j}\}_{\ell,j}$ and the vectors $\{L^c_{\ell,j}\}_{\ell,j}$. The simulator just chooses the set of bits $\{\chi^c_{\ell,j}\}_{\ell,j}$ uniformly at random, and also chooses the vectors $\{L^c_{\ell,j}\}_{\ell,j}$ at random from $d^{n \cdot v}$ under the constraint that $\sum_{\ell=1}^{n} \sum_{j=1}^{v} L^c_{\ell,j} = L^c$, and outputs these values. As the intermediate values $\{\chi^s_{\ell,j}\}_{\ell,j}$ are hidden from the distinguisher, the bits $\{\chi^c_{\ell,j}\}_{\ell,j}$ that the client receives from the invocations of Functionality 2.4 are distributed uniformly. Moreover, as the values $\{L^s_{\ell,j}\}_{\ell,j}$ are also hidden from the distinguisher, the vectors $\{L^c_{\ell,j}\}_{\ell,j}$ are all random under the constraint that they sum-up to the output of the client. The case of a corrupted server is proven analogously. ∎

Theorem 2.11 (Overall protocol)**.** *Protocol 2.8 realizes Functionality 2.1 against static corruptions in the semi-honest adversary model.*

Proof Sketch: As before, correctness is easy, and we need to show privacy. We note that except for the input and output values, the only thing beyond that the parties see in Protocol 2.8 are the vectors L^c, L^s that are returned by the intermediate Functionality 2.2, and that these vectors are individually uniform, irrespective of the input and output. Thus, in the case of a corrupted client the simulator just chooses L^c uniformly at random, and in the case of a corrupted server it chooses L^s uniformly at random. ∎

3 Breaking Sequences into Blocks

As stated in the Introduction the main idea underlying our solution is to approximate the edit distance between two sequences S and Q by partitioning both sequences into n blocks each, then summing up the edit distances across all blocks, returning $\sum_{\ell=1}^{n} \mathsf{ED}(Q_\ell, S_\ell)$. In this section we describe the method that we use to partition the sequences into blocks.

The idea of approximating the edit distance by computing the edit distance on small blocks is appealing as it yields an extremely efficient secure computation. However, the simplest manner of breaking the sequence/query into equal size blocks did not yield a good approximation of the edit distance over the full sequence. Thus, the question arose whether we can enable both parties to break their sequences into blocks that would also yield a good approximation of the edit distance.

3.1 Utilizing a Reference Genome

In order to refine the breaking into the blocks we introduced a commonly known reference genome R and had both the server and the client break their sequences in relation to R. The anticipation was that if we chose the reference genome R wisely, then it would aid in breaking the sequence and the query into blocks in a manner that would yield a better approximation of the full edit distance. The breaking of the sequences/query is done by computing an edit distance between the sequence/query and the reference genome. It might seem that we have accomplished nothing as we require a full computation of edit distance. However, the fact of the matter is that we off-load this computation to the server and client to be computed locally. The local computations are done in the clear and thus are much more efficient than computing them via a secure computation. Moreover, as we have seen, the preprocessing is re-usable, and for a large amount of queries this overhead becomes minor.

In our solutions we rely on a reference genome R of roughly the same length as the sequences that we want to break. To break a sequence S (or a query Q) into blocks, we run the Wagner-Fischer edit distance algorithm (for full details see Section A) to compute the edit distance between R and S. The algorithm also returns the PTR matrix that keeps the alignment between R and S. From the upper-left corner of the PTR to the lower-right corner it traces the path of how the minimum edit-distance can be obtained.

Let b be a parameter representing our desired block size (on the selection of b see Section 5). With S being recorded at the top of the matrix we break it as follows. We traverse the minimum edit distance path in PTR and whenever we have moved down b rows we break the sequence in that position into a block. Note, that the sizes of the blocks in this partition will vary. Most blocks are of size b, but some are shorter or longer. A full specification of the partitioning algorithm can be found in Algorithm 3.1.

Algorithm 3.1: $\mathsf{BreakToBlocks}_{R,\mathsf{b}}(S)$ **– Partition a Sequence into Blocks**

- **Parameters:** A reference sequence $R = (\rho_1, \ldots, \rho_r)$, block-size parameter b.
- **Input:** A sequence $S = (\sigma_1, \ldots, \sigma_s)$
  1. Invoke $\mathsf{ED}(R, S)$, and store the table PTR.
  2. Start at the top-left corner of PTR for each multiple of b, i.e. $\mathsf{b}, 2\mathsf{b}, \ldots$ find the index $j_1, j_2, \ldots$ such that $(i\mathsf{b}, j_i)$ is on the minimum edit-distance path. (If there is more than one pair for the same value $i \cdot \mathsf{b}$, store the index j that is closest to $i \cdot \mathsf{b}$.)
  3. Denote $n = \lceil r/\mathsf{b} \rceil$ (observe that the previous steps defines exactly $n-1$ indexes). Let $j_1, \ldots, j_{n-1}$ be the stored indexes, set $j_0 = 0$ and $j_n = s$. Define the blocks $S_\ell = (\sigma_{1+j_{\ell-1}}, \ldots, \sigma_{j_\ell})$ for every $1 \leq \ell \leq n$.
- **Output:** Output the blocks $S_1, \ldots, S_n$.

Experimental observations. A major factor in both accuracy and the performance is the number of distinct values for each block (i.e., $|Q_\ell|$). If the number of distinct blocks is small for large b, then we can save a lot in running time, as n (the number of blocks) is smaller, while still the work that we have to spend per block is relatively small. For all our datasets, the number of distinct values in a block was no more than 10 in all datasets, even for block size $\mathsf{b} = 8$. We elaborate on that in Section 4.2.

4 Accuracy of Our Approximation

We examine the accuracy of our approximation both theoretically and empirically. In Section 4.1 we provide a theoretical analysis of our approximation algorithm. While the bounds in this analysis are somewhat coarse, the analysis still shows that the algorithm is accurate if the reference genome is "close" to the database, or when all sequences in the database are equally-far from the reference genome. In Section 4.2, we show an empirical evaluation of our algorithm on real genomic datasets.

4.1 Theoretical Analysis

Intuition for accuracy. The following disregards the case of a "block miss", in which we discuss in Appendix B. That is, we focus now on the quality of approximating the edit-distance between two sequences X and Y by computing block-wise distances, where the partitioning into block is performed with respect to a common reference genome as described in Algorithm 3.1.

In a more detail, suppose we have two sequences X and Y and we wish to compute their edit-distance $\mathsf{ED}(X,Y)$. Our approximation algorithm first breaks each one of the sequences into blocks $X_1,\ldots,X_n$ and $Y_1,\ldots,Y_n$. Then, it computes

$$\mathsf{ApproxED}(X,Y) = \sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, Y_\ell) \ .$$

First, it is clear that

$$\mathsf{ED}(X,Y) \leq \mathsf{ApproxED}(X,Y) = \sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, Y_\ell) \ .$$

This is because there are many ways one can "transform" sequence X into sequence Y. In our case, we transform each block of X into the corresponding block of Y, and sum the number of operations these block-wise transformation consumes. The term $\mathsf{ED}(X,Y)$ minimizes over all possible ways to transforms X into Y, including that specific aforementioned possibility.

Let $R = (R_1,\ldots,R_n)$ be the strings of the reference genomes, after breaking it into b-size blocks. For every $\ell \in \{1,\ldots,n\}$, it holds that

$$\mathsf{ED}(X_\ell, Y_\ell) \leq \mathsf{ED}(X_\ell, R_\ell) + \mathsf{ED}(R_\ell, Y_\ell) \ .$$

This holds from a similar reasoning as before: There are many ways to transform X_ℓ into Y_ℓ. The optimal way consumes $\mathsf{ED}(X_\ell, Y_\ell)$ operations, whereas the right hand-side is just one possible way – transforming X_ℓ into R_ℓ, and then transforming R_ℓ into Y_ℓ.

The way we choose the partitioning of the sequences X and Y into blocks relies on the full edit distances $\mathsf{ED}(X,R)$ and $\mathsf{ED}(Y,R)$, which implies that the following *equations* hold:

$$\mathsf{ED}(X,R) = \sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, R_\ell), \quad \text{and} \quad \mathsf{ED}(Y,R) = \sum_{\ell=1}^{n} \mathsf{ED}(Y_\ell, R_\ell) \ .$$

Putting it all together, since $\sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, Y_\ell) = \sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, R_\ell) + \mathsf{ED}(R_\ell, Y_\ell) = \mathsf{ED}(X,R) + \mathsf{ED}(Y,R)$, we conclude the following upper bound:

$$\mathsf{ED}(X,Y) \leq \mathsf{ApproxED}(X,Y) \leq \mathsf{ED}(X,R) + \mathsf{ED}(Y,R) \ .$$

While this bound is coarse, still it provides some meaningful insights:

- First, if the reference genome has relatively the same distance from X and Y as any other sequence in the database (i.e., all values $\mathsf{ED}(X,Y)$, $\mathsf{ED}(X,R)$ and $\mathsf{ED}(Y,R)$ are similar), this is 2-approximation.
- Second, if the two sequences X and Y are obtained by adding random mutations to the reference genome in different distinct locations, then this approximation is in fact, exact.

4.2 Empirical Evaluation

We empirically evaluate the accuracy of our approximation protocol. We specifically target "high-divergence" regions of the genome, so we seek to verify that we still get good results even for such regions. We tested our approach on various datasets and on different chromosomes:

Our main dataset: ZNF717. Our main dataset was provided by the organizers of the iDash competition [iDA16]. It contains relatively many (501) gene sequences extracted from the publicly

Coding Region Gene	Location	# Samples	Length	Avg. Δ (stdev)	Max-Δ Pair	Max-Δ Ref	Variability
ZNF717	chr 3:75785026–75788496	501	3470	91.33 (94.24)	175	184	$\leq 5.04\%$
TEKT4P2	chr 21:9907190–9909277	101	2087	27.49 (28.55)	54	57	$\leq 2.58\%$
CDC27P1	chr 2:133019901–133020615	101	714	12.41 (13.42)	33	50	$\leq 4.62\%$
CDC27P2	chr Y:10027986-10029907	101	1950	29.31 (30.46)	65	68	$\leq 3.33\%$
ABHD17AP5	chr 22:22720578–22722138	15	1570	3.01 (3.51)	6	18	$\leq 0.38\%$
Simulated		10000	3470	173.0 (26.14)	350	158*	$\leq 10\%$

Table 1: Our datasets. Avg. Δ is the average of all edit distances between pairs in the datasets. Max-Δ Pair – is the maximal edit distance among all pairs in the dataset. Max Δ-Ref – is the maximal distance between a sequence and the reference genome. Variability is the maximal distance divided by the size of the region. In the simulated database, the distance is from a synthetic reference genome.

available 1000 Genomes Project [Int16]. It was extracted from human chromosome 3 (75785026-75788496), of length just under 3500, within the coding region of gene ZNF717. The iDASH organizers explained the choice of this particular gene by its high divergence among individual genomes.

Other datasets. From a set of 170 complete sequences, and with the help of the iDash organizers, we extracted several other regions with high-divergence. For each region we chose only a subset of the samples (usually 101) to make the task more challenging. That is, we excluded samples that were identical to each other within that region. The following datasets were extracted:

- **TEKT4P2:** Chromosome 21 (9907190–9909277) of size under 2100 within the coding region of gene TEKT4P2.
- **CDC27P1:** Chromosome 2 (133019901–133020615) of size under 750 within the coding region of gene CDC27P1 (known also as ZN.
- **CDC27P2:** Chromosome Y (10027986-10029907) of size under 1950, within the coding region of gene CDC27P2.
- **ABHD17AP5:** Chromosome 22 (22720578–22722138) of size under 1570, within the coding region of gene ABHD17AP5. Here the region has very low variability. We therefore extracted only 15 samples, testing our algorithm also for a "toy" database.

The datasets, including some basic properties are given in Table 1. In Appendix C we provide some more details about each dataset. We present the distribution of distances between pairs of sequences in each database, and also show the distribution of distances of some random queries to the other sequences in the perspective databases. Overall, these distributions show a nice variety in the type of queries we examined. Sometimes, there are queries in which the set of k closest sequences is easily recognizable, as the distance between the $k+1$'th closest element and the query is significantly greater than the distance between the query and the k'th closest element. In most cases, however, these two distances are very small or even identical, making the set of k closest sequences harder to recognize.

Accuracy. The main results of our accuracy test are summarized Table 2. Our algorithm performs remarkably well on all tested datasets, where it returned the exact result in almost all tests, and a very close results otherwise (most of the cases, a result with the same edit distance as the edit distance of the k'th element, or one farther).

We ran the following experiment for each one of the datasets: We chose $\approx 10\%$ random sequences from the dataset as queries and the rest of the datasets was set to be the database. We ran the preprocessing phase of our protocol, and compared the set of sequences that the protocol returned

to the correct values, for different threshold parameters – $k = 1, 3, 5$ and 10 (that is, finding the closest sequence, the set of three closest sequences, etc.). We repeated the experiments 10 times, for independent random choice of queries. The block size was set to $\mathsf{b} = 3$, while similar results are obtained to other choices of this parameter.

Dataset	k	Average ED (std)	Average-Δ (std)	Precision
ZNF717	1	2.07 (5.27)	0 (0)	100%
ZNF717	3	2.96 (5.39)	0 (0)	100%
ZNF717	5	4.68 (9.59)	0.01 (0.1)	98.85%
ZNF717	10	28.98 (27.46)	0.25 (0.82)	97.48%
TEKT4P2	1	13.14 (3.38)	0 (0)	100%
TEKT4P2	3	16.29 (2.77)	0.80 (4.30)	96.66%
TEKT4P2	5	18.5 (2.5)	0.73 (3.95)	96.66%
TEKT4P2	10	21.39 (4.54)	0.60 (3.23)	97.33%
CDC27P1	1	2.81 (1.83)	0.02 (0.14)	95.91%
CDC27P1	3	4.39 (1.78)	0.18 (0.74)	94.56%
CDC27P1	5	5.47 (2.38)	0.33 (0.93)	94.28%
CDC27P1	10	6.87 (2.45)	0.57 (1.48)	96.94%
CDC27P2	1	13.08 (4.18)	0 (0)	100%
CDC27P2	3	16.75 (3.74)	0 (0)	100%
CDC27P2	5	18.27 (3.75)	0.03 (0.26)	99.67%
CDC27P2	10	20.55 (3.58)	0 (0)	99.67%
ABHD17AP5	1	0.92 (0.76)	0 (0)	100%
ABHD17AP5	3	2.75 (1.3)	0.6 (0.8)	86.67%
ABHD17AP5	5	3.17 (1.21)	0.2 (0.4)	92%
ABHD17AP5	10	4.92 (1.03)	0 (0)	98%

Table 2: Accuracy of our algorithm, for the various datasets and various choices of k.

Table 2 summarizes the accuracy results of our approximation algorithm in the different datasets for different k. The table consists of the following columns:

- Dataset.
- k is the threshold parameter – how many sequences to return.
- Average ED (std) is the (true) average edit distance between the query and the set of the kth closest sequence in the database. The value in parenthesis is the standard deviation.
- Precision: Among the set of the true k closest elements, how many elements did the approximation algorithm return. We break ties according to the lexicographically order. This implies that in case of a tie in which the approximation algorithm returned a sequence with the right edit distance but greater id than the lexicographically smallest one, we count it as an error.
- Average-Δ (std) is the average of how much farther the farthest record returned by the algorithm than the k'th-closest record.

The ZNF717 dataset. The dataset in which we had the most number of samples, as well the most various type of queries is the dataset ZNF717. We report some more detailed results in that datasets.

We observe that the block-size parameter does not effect much the accuracy of the algorithm. Nevertheless, it does effect the performance of the protocol, as larger block size means less blocks

to process, and therefore overall less workload. Table 3 summarizes the performance and accuracy results of our approximation algorithm as a function of the blocksize parameter b, when returning the closests 5 (approximate) distances.

Block Size (b)	b'	# Values	Average-Δ (std)	Average Rank (std)	Max-K
3	10	6	0 (0)	5 (0)	6
5	12	8	0.01 (0.1)	5.01 (0.1)	9
8	15	10	0 (0)	5 (0)	7
12	19	10	0.01 (0.1)	5.01 (0.14)	9

Table 3: Algorithm accuracy as function of block size b. Average(stdev) edit-distance between query and 5'th closest sequence is 4.15(8.37).

The table consists of the following columns:

- $\underline{\mathsf{b}'}$ is the largest actual blocksize obtained for any of the sequences (i.e., after breaking the sequences into blocks, some blocks can be larger than b.)
- # Values is the largest number of distinct values found in any block (i.e., the parameter v).
- Average-Δ is how much farther is the farthest record returned by the algorithm than the true k'th-closest record.
- Average-rank is the true rank of the farthest result returned.
- Max-K is how many records we should have returned to ensure that the true k'th closest record is always part of the result set.

In this experiment, we chose $\approx 1\% \approx 5$ sequences as query sequences, and the other records were chosen to be the database. We repeated this choice 100 times, and the "Average-$\star$" values are computed over these 100 runs, together with the standard deviation (in parenthesis).

The parameters b' and # values. Our overall aim is to compute edit-distance in genomic setting with higher efficiency. Besides exploring the accuracy of our algorithm, we also wish to explore the values of b' and # values as these two parameters play a pivotal role in the efficiency of our protocol. The parameter b' is important when realizing Functionality 2.3 using Yao's circuit, as the input size of the parties would be a bound on the maximal block size that might appear. The parameter # values is the value v in the protocol, and reflects the amount of times we will invoke the underlying subprotocols. In the following table, we show how these two parameters appear in the datasets. The experiment is the same as in Table 2.

	$\mathsf{b}=5$		$\mathsf{b}=8$	
Dataset	b'	Max # values	b'	Max # values
ZNF717	12	8	15	10
TEKT4P2	5	7	8	10
CDC27P1	7	4	12	4
CDC27P2	8	4	12	4
ABHD17AP5	5	2	8	2

Table 4: Parameters b' and # values per dataset and b.

DB Size		Max-rank of any returned record 5	6	7	8	≥ 9	MaxK (Rank)	Real ED (std)
500	%Q	68	20	9	3	0	10 (8)	139.08 (27.39)
	ΔED	0(0)	1.00(0)	1.11(0.56)	1(0)	0 (0)		
1000	%Q	72	16	5	4	3	19 (18)	136.29 (27.25)
	ΔED	0(0)	1.06(0.24)	1(0)	1.5(0.5)	2.33 (0.47)		
2000	%Q	65	22	9	2	2	20 (12)	135.49 (27.19)
	ΔED	0(0)	1.05(0.2)	1.11(0.32)	1(0)	1.5 (0.5)		
4000	%Q	67	18	10	2	3	14 (11)	134.47 (27.2)
	ΔED	0(0)	1(0)	1.10(0.3)	1(0)	1 (0)		
8000	%Q	76	10	6	4	4	12 (10)	133.57 (27.15)
	ΔED	0(0)	1(0)	1(0)	1.25(0.43)	1.25 (0.43)		

Table 5: Accuracy of our algorithm, for varying DB sizes. Each row represents the results of our algorithm with synthetic reference genome, $\mathsf{b} = 3$ and $k = 5$.

4.3 Simulated Dataset

We wanted to test the scalability of our secure protocol when processing databases with many more records, with the aim of verifying that the algorithm is accurate also with large databases. Due to the lack of availability of such data a larger dataset of 10,000 sequences (of length around 3470) was generated via simulation by the organizers of the iDash competition. It had a significantly higher divergence than the real datasets, specifically, the maximal distance between a pair of records might be $\approx 10\%$ of the region, compared to maximal distance of at most 5% in the real datasets. Furthermore, it no longer related to the public reference genome as the real data did, as it was not real genetic material. Thus, we also manually created a "fake" (synthesized) reference genome that related to the synthesized database in the same manner that the public reference genome relates to real genetic datasets. In our experiments with the synthesized data we used the fake reference genome.

Table 5 details the accuracy of our algorithm as a function of the database size (DB Size) for the synthetic dataset, blocksize parameter $\mathsf{b} = 3$, and when returning $k = 5$ records. The reference genome R is the synthesized reference genome. Similar results are obtained for other block sizes as well. The table consists of the following columns:

- $\underline{\%Q}$ the distribution of the true rank of the k'th element in the result returned. For all DB size, at least 96% of the queries have rank at most 8.
- $\underline{\Delta\text{ED}}$ is the average of how much farther the farthest recored returned by the algorithm than the true k'th-closest record. The average is taken among the queries of the corresponding rank $r \in \{5, \ldots, 8\}$ or ≥ 9. The value in parenthesis is the standard deviation.
- Max-K is how many records we should have returned to ensure that the true k'th closest record is always part of the result set. The (Rank) values in parenthesis are the maximum true rank of any record in the result set.
- Real ED is the average (over all queries) of the accurate edit-distance between the query and the true k'th-closest database record.

DB Size	$v =$ # values	Preprocessing (s)	Query			Bandwidth (MB)	# AND-gates
			Compare (s)	OTs (s)	k-min (s)		
500	15	14.9	0.9	1.4	0.05	80	789595
1000	25	30	1.51	4.36	0.16	180	1399480
2000	30	61.8	2.1	11.7	0.31	340	2035415
4000	35	119	2.8	28.2	0.6	660	3149350

Table 6: Running times for varying DBs. In all executions, we also produce a synthetic reference genome, which takes 7.6 seconds to produce, and should be added to the preprocessing time.

5 Evaluation and Performance

The real dataset for ZNF717 has sequences that are fairly close together and also close to the reference genome. Specifically, we had a database with $m = 500$ records of size less than $w = 3500$, and edit distances below $d = 256$, but the closest five to every query were always within edit distance of less than 50. Also we kept the block size parameter at a low $\mathsf{b} = 3$, and as a result the sizes of blocks in all our sequences were always of length below 4 (but we used a larger $\mathsf{b}' = 16$ to ensure that there are no errors) and the number of distinct values per block was bounded below 8 (but here too we used $v = 16$ to ensure that there are no errors).

To stress-test our approach we did most of our testing on synthetic data that shows larger variability. We received from the iDASH organizers 10,000 synthetic sequences, we chose 100 of them to be query sequences and $m = 500, 1000, 2000$ or 4000 of them[4] to form databases of difference sizes. In all these cases, the block-sizes that we obtained were bounded by 4 and the number of distinct values per block was at most 35 even for the largest database (here we used the bounds $\mathsf{b}' = 16$ and $v = 35$ in the protocol).

We implemented our protocol over the `C++` version of the Secure Computation API library (SCAPI) [EFLL12]. We use the state-of-the-art improvements, include Yao with free-XOR technique [KS08] and half-gates [ZRE15], and the recent improvements in OT-extension [ALSZ13, KOS15]. Table 6 presents the performance results for varying database size, these numbers were obtained by running the protocol on a single x86_64 machine using the loopback device for client-server communication.

In our implementation, the most costly aspect was the pre-processing on the server side (which only needs to be done once per database). This part requires many edit distance computations (in the clear), and we did not attempt to optimize it. In the online time, the most expensive part are the OT protocols.

In Table 6, we report the maximal allowed size of the tables (# values, i.e., v in the protocol), the times took for the preprocessing, answering a query, the bandwidth and the number of AND-gates.

On the choice of parameter b. In terms of accuracy, we observe similar results for different block sizes. Nevertheless, in terms of efficiency, it seems more appealing to run with larger block sizes b. Specifically, recall that the running time is proportional to $n \cdot v$, where n is the number of blocks and v is the size of $|T_\ell|$ (recall that this is the number of indicator bits that we compute, as well as the number of strings $L_{\ell,j}$ that the parties share). Thus, larger block size b implies smaller n.

[4]Unlike accuracy, in our secure protocol we stopped with a database of size 4000 due to some technical problems with our implementation, and in principal the running times should scale similarly to larger datasets. We remark that our protocol implementation was also tested and evaluated by external referees as part of our participation in the iDash competition, and similar running times were confirmed.

This is true as long as the size of the tables v remains small. For the real database, v is small even for $\mathsf{b} = 12$, on the other hand, for the synthesized database, we run mainly with $\mathsf{b} = 3$. Of course, the server can locally compute the parameter v for several different block sizes b, and chose the one that minimizes $n \cdot v$ (and sends the client the parameters b and v at the beginning of the protocol).

6 Conclusions

In this work we described a privacy preserving protocol for answering Similar Patient Queries (SPQ) on genome data. Our protocol was designed to operate in settings with high divergence between individuals, where a previous solution due to Wang et al. [WHZ+15] does not apply. We developed an efficient method for approximating the edit distance that provides very good accuracy even in regions of the genome with ≈5% variability, while at the same time being 2-3 orders of magnitude faster than exact calculation.

Our work was motivated by the 2016 iDASH competition for competing on genome data In particular, for the 500-record dataset used in that competition, we can answer SPQ in under two seconds per query (after about 15 seconds of one-time pre-processing of the database). We believe that this solution is applicable in real-life situations where SPQ on remote genome data is desirable, and plan to peruse real-world uses of it in the future.

Acknowledgment

We thank Shalev Keren, Meital Levy and Assi Barak for the implementation of our protocol. We thank Diyue Bu and Haixu Tang for their support and help in extracting the datasets, and Haixu Tang, Diyue Bu, XiaoFeng Wang, Shuang Wang, Xiaoqian Jiang, and Lei Wang for organizing the iDASH competition and helping us with all our questions and requests. We also than Robin Hui for the proof in Section 4.1.

References

[ABOcS15] Mete Akgün, A. Osman Bayrak, Bugra Ozer, and M. Şamil Sağıroğlu. Privacy preserving processing of genomic data: A survey. *Journal of Biomedical Informatics*, 56:103 – 111, 2015.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Conference on Computer and Communications Security*, pages 535–548. ACM, 2013.

[AO12] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012.

[BBC+11] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 691–702, 2011.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM*

on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015, pages 51–58, 2015.

[EFLL12] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. SCAPI: the secure computation application programming interface. *IACR Cryptology ePrint Archive*, 2012:629, 2012. A link to the library: `http://crypto.biu.ac.il/about-scapi`.

[FIM+01] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. In *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 927–938. Springer, 2001.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *ACM Symposium on Theory of Computing, STOC*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*, 2011.

[HSE+11] Yan Huang, Chih-Hao Shen, David Evans, Jonathan Katz, and Abhi Shelat. Efficient secure computation with garbled circuits. In *Information Systems Security - 7th International Conference, ICISS 2011, Kolkata, India, December 15-19, 2011, Procedings*, pages 28–48, 2011.

[iDA16] iDASH - integrating Data for Analysis, Anonimization, and SHaring, 2016. Webpage at `https://idash.ucsd.edu/genomics`, 2016 competition at `http://www.humangenomeprivacy.org/2016/`.

[Int16] International Genome Sample Resource. IGSR and the 1000 genomes project. `http://www.internationalgenome.org/`, Accessed Nov 2016.

[JKS08] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 216–230, 2008.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology - CRYPTO*, pages 724–741, 2015.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming, 35th International Colloquium, ICALP*, pages 486–498, 2008.

[NAC+15] Muhammad Naveed, Erman Ayday, Ellen W Clayton, Jacques Fellay, Carl A Gunter, Jean-Pierre Hubaux, Bradley A Malin, and XiaoFeng Wang. Privacy in the genomic era. *ACM Computing Surveys (CSUR)*, 2015.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1), January 1974.

[WHZ+15] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 492–503, New York, NY, USA, 2015. ACM.

[Wik17] Wikipedia. Reference genome — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Reference_genome`, 2017. [Online; accessed 19-May-2017].

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *Symposium on Foundations of Computer Science, FOCS*, pages 162–167, 1986.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT*, pages 220–250, 2015.

A The Wagner-Fischer Algorithm

The WF algorithm [WF74] is based on dynamic programming, for computing the edit distance between two sequences $A = (\alpha_1, \ldots, \alpha_a)$ and $B = (\beta_1, \ldots, \beta_b)$.

The algorithm proceeds by preparing an $(a+1)$-by-$(b+1)$ matrix $D[\cdot,\cdot]$, where entry (i,j) is the edit-distance between the i-prefix of A and the j-prefix of B. The first row and column are initialized by $D[i,0] = i$, $D[0,j] = j$ for all $0 \le i \le a$ and $0 \le j \le b$. Then for $1 \le i \le a$ and $1 \le j \le b$ the algorithm iteratively sets

$$D[i,j] = \begin{cases} \text{if } \alpha_i = \beta_j: & D[i-1,j-1] \quad \text{(match)} \\ \text{otherwise}: & \min \begin{cases} D[i-1,j-1]+1 & \text{(substitution)} \\ D[i-1,\ j]+1 & \text{(delete)} \\ D[i\ ,j-1]+1 & \text{(insert)} \end{cases} \end{cases}$$

and finally it returns the answer $D[a,b]$.

This procedure can be augmented to return not only the edit distance itself but also the sequence of operations that transforms A to B in $D[a,b]$ steps. Specifically, together with D we also prepare a matrix of pointers $PTR[\cdot,\cdot]$ (with the same dimension as D), that for each entry (i,j) points to the previous entry from which $D[i,j]$ received its value. Specifically, we initialize $PTR[0,0] = \perp$, $PTR[i,0] = (i-1,0)$ for all $1 \le i \le a$ and $PTR[0,j] = (0,j-1)$ for all $1 \le j \le b$, and then for $1 \le i \le a$ and $1 \le j \le b$ we iteratively set

$$PTR[i,j] = \begin{cases} (i-1,j-1) & \text{if } D[i,j] \le D[i-1,j-1]+1 \\ (i-1,\ j) & \text{if } D[i,j] = D[i-1,j]+1 \\ (i\ ,j-1) & \text{if } D[i,j] = D[i,j-1]+1 \end{cases}$$

where the first case corresponds to a match or substitution, the second corresponds to a delete, and the last case corresponds to an insert. When more than one condition applies, we break ties toward the main diagonal. Namely, we prefer $(i\ ,j-1)$ to the other options when $j > i$, prefer $(i-1,\ j)$ when $i > j$, and prefer $(i-1,j-1)$ when $i = j$.

The PTR table lets us trace on optimal path, starting from $PTR[a,b]$ and following the pointers to get both the alignment of the sequences A, B, as well as the corresponding operations (match, substitute, insert, delete).

B Block Misses

Our approach ignores blocks of the client that do not appear in the database. This introduces some error to our approximation, but we empirically verify that it is minor. First, in Section B.1 we study the frequency of block-misses and see that they rarely occur. Second, in Section B.2 we study the error that is introduced with each such block-miss, and see that it is indeed very small.

B.1 Empirical Evaluation: Frequency of Block-Misses

Our approach ignores blocks of the clients that do not appear in the database. Intuitively, assuming that m independent samples from some distribution (i.e., the ℓth block of each one of the sequences) occur in some very small set (i.e., T_ℓ), then the probability that an additional independent sample (i.e., Q_ℓ) will occur in the same set is close to 1.

In Table 7, we report the frequency of block misses and show that it is a relatively rare event. The data below is on the real database (ZNF717), where we chose 30 sequences at random to be the queries, and the other 470 sequences to be the DB. We then built the tables T_ℓ, and run our protocol. Overall, more than 99.95% of the blocks of the queries do appear as one of the blocks in the DB. In fact, for more than half of the queries, all their blocks appear in the DB, and the maximal number of block misses per query that was observed is 3. In Table 7 we observe similar results even for small databases, and even when the number of queries and the number of sequences in the database are close.

DB size	#Queries	Average # hits per query (STD)	Max # of misses per query
276	224	1155.91 (0.62)	2
340	160	1155.90 (0.61)	3
400	100	1155.88 (0.62)	3
434	66	1155.95 (0.49)	2
470	30	1156.51 (0.66)	2

Table 7: Frequency of number of block of the queries Q_ℓ that appear or do not appear in the corresponding set T_ℓ of the DB, as a function of the DB size. The DB is the real database, where random number of elements where chosen to be the DB and the rest where chosen to be the queries. Block size b is always 3, and so the number of blocks is always 1157.

B.2 On the Error Introduced by Block Miss

Even though that block misses are relatively rare events, it is still a question what to do in case they occurs. Assume that $Q_\ell \notin T_\ell$ for some block $\ell \in \{1, \ldots, n\}$. In the following, we compare between two possible approaches:

- The first approach is to compute the accurate distance between $\mathsf{ED}(Q_\ell, S_{i,\ell})$ for every $S_{i,1}, \ldots, S_{i,m}$. This introduces some additional complexity to the protocol, as we have to hide, both to the client and to the server, on which blocks Q_ℓ it holds that $Q_\ell \notin T_\ell$, as well as to compute edit-distances (of small blocks) in the online time. The resulting approximation function is as follows: $\mathsf{ApproxED}(Q, S_i) = \sum_{\ell=1}^{n} \Delta(Q_\ell, S_{i,\ell})$ where,

$$\Delta(Q_\ell, S_{i,\ell}) \quad = \quad \mathsf{ED}(Q_\ell, S_{i,\ell}) \ . \tag{B.1}$$

- The second approach is the one we chose: we simply ignore these blocks. This results in the following approximation function (as we have already seen in Section 2.1):
  $\mathsf{ApproxED}(Q, S_i) = \sum_{\ell=1}^{n} \Delta(Q_\ell, S_{i,\ell})$, where

$$\Delta(Q_\ell, S_{i,\ell}) \quad = \quad \begin{cases} \mathsf{ED}(Q_\ell, S_{i,\ell}) & \text{if } Q_\ell \in T_\ell \\ 0 & \text{otherwise} \end{cases} . \tag{B.2}$$

In Table 8, we show that the accuracy improvement is minor when choosing the first approach. This justifies our choice, as the overhead in computation for computing full edit-distances in case of block misses is significant. The experiment is the same as in Table 2, with the same setting and same random choices of queries.

	Approach I: Eq. (B.1)		Approach II: Eq. (B.2)	
Dataset	Average-Δ (std)	Precision	Average-Δ (std)	Precision
ZNF717	0.25 (0.82)	96.44%	0.25 (0.82)	97.48%
TEKT4P2	0.54 (2.59)	99.28%	0.60 (3.23)	97.33%
CDC27P1	0.15 (0.69)	97.66%	0.57 (1.48)	96.94%
CDC27P2	1.16 (4.48)	99.02%	0 (0)	99.67%
ABHD17AP5	0 (0)	100%	0 (0)	98%

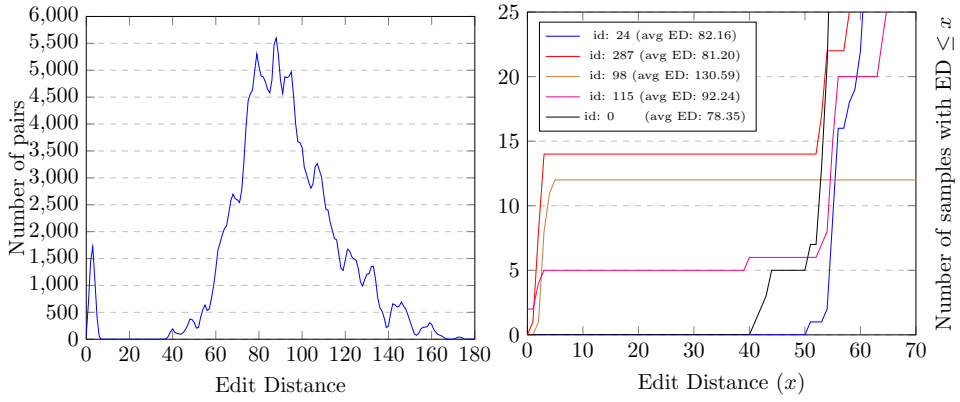
Table 8: Accuracy loss for block-misses. Comparing between Approach I: computing $\Delta(Q_\ell, S_{i,\ell}) = \mathsf{ED}(Q_\ell, S_{i,\ell})$ in case $Q_\ell \notin T_\ell$ (as in Eq. (B.1)), and Approach II: $\Delta(Q_\ell, S_{i,\ell}) = 0$ in case $Q_\ell \notin T_\ell$ (as in Eq. (B.2)). The datasets, experiments and random choices are the same as in Table 2, for $k = 10$.

C The Datasets

We show some properties regarding the distribution of edit-distances in the tested datasets. For each one of the datasets, we provide two graphs:

- First, we show the distribution of distances between pairs of sequences. All datasets share relatively similar properties, whereas the dataset of ZNF717 contains several queries that are "close" to each other, and some query that are "far" than any other sequence in the database.
- Second, for each dataset we picked several random sequences, and show the distribution of their distances to other sequences in the database.

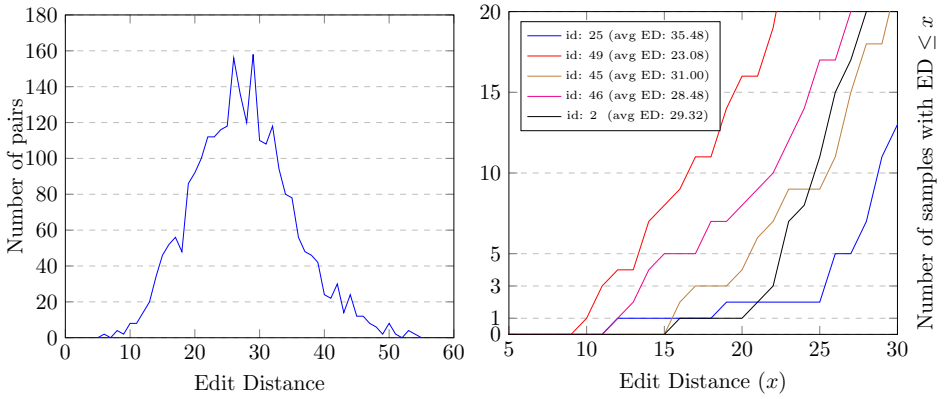
The graphs of distribution shows the variety of queries our algorithm successfully deals with. For some queries the set of k closest sequences is easily recognizable, as the distance between the $k+1$'th closest element and the query is greater than the distance between the query and the k'th closest element. However, some queries our more challenging, as there are many sequences in the database that are very close to the set of k closest queries. Any mistake in approximating each one of them might result in an incorrect returning set.



(a) Distribution of pairwise distances in the dataset. Number of pairs with edit distance x.

(b) 5 random queries – number of elements in the database with edit distance $\leq x$.

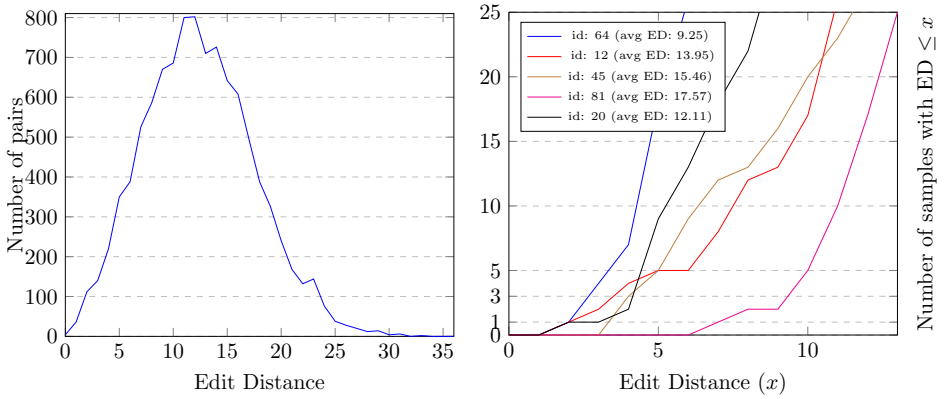
Figure 1: Dataset ZNF717: (3:75785026–75788496). Dataset of 501 samples of size ≈ 3470. Average distance between pair: 91.33 (stdev: 94.24). Maximal distance: 175.



(a) Distribution of pairwise distances in the dataset. Number of pairs with edit distance x.

(b) 5 random queries – number of elements in the database with edit distance $\leq x$.

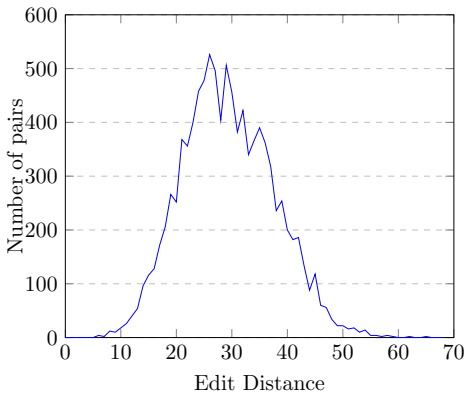
Figure 2: Dataset TEKT4P2: (21:9907190-9909277). Dataset of 101 samples, of size ≈ 2087. Average distance between pair: 27.49 (stdev: 28.55). Maximal distance between pair: 54.



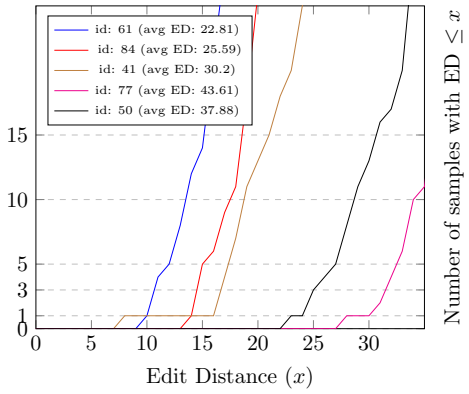
(a) Distribution of pairwise distances in the dataset. Number of pairs with edit distance x.

(b) 5 random queries – number of elements in the database with edit distance $\leq x$.

Figure 3: Dataset CDC27P1:(2:133019901-133020615). Dataset of 101 samples, of size ≈ 714. Average distance between pair: 12.41 (stdev: 13.42). Maximal distance: 33.

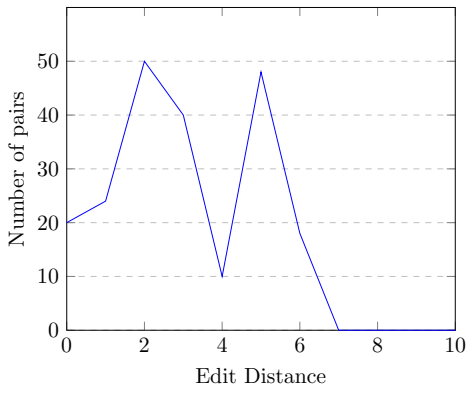


(a) Distribution of pairwise distances in the dataset. Number of pairs with edit distance x.

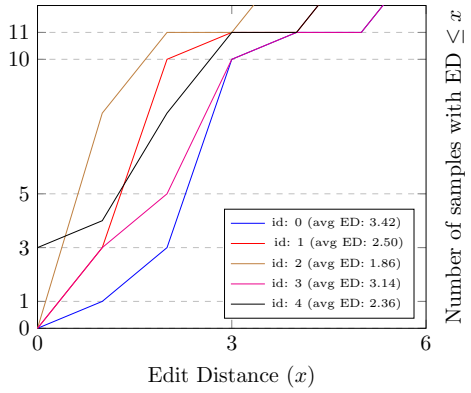


(b) 5 random queries – number of elements in the database with edit distance $\leq x$.

Figure 4: Dataset CDC27P2: (Y:10027986-10029907). Dataset of 101 samples, of size ≈ 1950. Average distance between pair: 29.31 (stdev: 30.46). Maximal distance: 65.



(a) Distribution of pairwise distances in the dataset. Number of pairs with edit distance x.



(b) 5 random queries – number of elements in the database with edit distance $\leq x$.

Figure 5: Dataset ABHD17AP5: (22:22720578-22722138). Dataset of 15 samples, of size ≈ 1570. Average distance between pair: 3.01 (stdev: 3.51). Maximal distance: 6.