Published in final edited form as:
IEEE Trans Inf Theory. 2021 June ; 67(6): 3295–3314. doi:10.1109/tit.2020.3030569.

Trace Reconstruction Problems in Computational Biology

Vinnu Bhardwaj[1], **Pavel A. Pevzner**[2,*], **Cyrus Rashtchian**[2,3], **Yana Safonova**[2]

[1]Electrical and Computer Engineering Department, University of California San Diego, La Jolla, USA.

[2]Computer Science and Engineering Department, University of California San Diego, La Jolla, USA.

[3]Qualcomm Institute, University of California San Diego, La Jolla, USA.

Abstract

The problem of reconstructing a string from its error-prone copies, *the trace reconstruction problem*, was introduced by Vladimir Levenshtein two decades ago. While there has been considerable theoretical work on trace reconstruction, practical solutions have only recently started to emerge in the context of two rapidly developing research areas: immunogenomics and DNA data storage. In immunogenomics, traces correspond to mutated copies of genes, with mutations generated naturally by the adaptive immune system. In DNA data storage, traces correspond to noisy copies of DNA molecules that encode digital data, with errors being artifacts of the data retrieval process. In this paper, we introduce several new trace generation models and open questions relevant to trace reconstruction for immunogenomics and DNA data storage, survey theoretical results on trace reconstruction, and highlight their connections to computational biology. Throughout, we discuss the applicability and shortcomings of known solutions and suggest future research directions.

I. Introduction

TWO decades ago, Vladimir Levenshtein introduced the Trace Reconstruction Problem, reconstructing an unknown *seed* string from a set of its error-prone copies, which are referred to as *traces* [1]. In information-theoretic terminology, the seed string is observed by passing it through a noisy channel multiple times. Levenshtein set forth the challenge of developing efficient algorithms to infer the seed string and characterizing the number of traces needed for its reconstruction [2], [3]. He succeeded in solving these problems in the case of the *substitution channel*, where random symbols in the seed string are mutated independently, and demonstrated that a small number of deletions or insertions may be tolerated. A few years later, Batu et al. [4] analyzed the trace reconstruction problem in the *deletion channel*, where random symbols are deleted from the seed string independently so that a trace is a random subsequence of the seed string.

[*] Corresponding author: ppevzner@ucsd.edu.

After these seminal papers [1]-[4], trace reconstruction has received a lot of attention, especially in the last few years [5]-[20]. However, despite a wealth of theoretical work, there is a surprising lack of practical trace reconstruction algorithms. Although Batu et al., [4] and many follow-up studies motivated trace reconstruction by the *multiple alignment problem* in computational biology [21], we are not aware of any software tools that use trace reconstruction for constructing multiple alignments and applying them for follow-up biological analysis.

Transforming a biological problem into a well-defined algorithmic problem comes with many challenges. An attempt to model all aspects of a biological problem often results in an intractable algorithmic problem while ignoring some of its aspects (like in the initial formulation of the Trace Reconstruction Problem) may lead to a solution that is inadequate for practical applications. Computational biologists try to find a balance between these two extremes and typically use a simplified (albeit inadequate) problem formulation to develop algorithmic ideas that eventually lead to practical (albeit approximate) solutions of a more complex biological problem.

The first applications of trace reconstruction emerged only recently in the context of two rapidly developing research areas: personalized immunogenomics [22], [23] and DNA data storage [24]-[33]. In this survey paper, we identify a variety of open trace reconstruction problems motivated by immunogenomics and DNA data storage, describe several practically motivated objectives for trace reconstruction, and discuss the applicability and shortcomings of known solutions. Our goal is to introduce information theory experts to emerging practical applications of trace reconstruction, and, at the same time, introduce computational biology experts to recent theoretical results in trace reconstruction.

A. Trace Reconstruction in Computational Immunology

How have we survived an evolutionary arms race with pathogens?: Humans are constantly attacked by pathogens that reproduce at a much faster rate than humans do. How have we survived an evolutionary arms race with pathogens that evolve a thousand times faster than us?

All vertebrates have an *adaptive immune system* that uses the *VDJ recombination* to develop a defensive response against pathogens at the time-scale at which they evolve. It generates a virtually unlimited variety of *antibodies*, proteins that recognize a specific foreign agent (called *antigen*), bind to it, and eventually neutralize it. There are $\approx 10^8$ antibodies circulating in a human body at any given moment (unique for each individual!) and this set of antibodies is constantly changing. How can a human genome (only $\approx 20,000$ genes) generate such a diverse defense system?

VDJ recombination: In 1987, Susumu Tonegawa received the Nobel Prize for the discovery of the VDJ recombination [34]. The *immunoglobulin locus* is a 1.25 million-nucleotide long region in the human genome that contains three sets of short segments known as *V, D*, and *J genes* (40 V, 27 D, and 6 J genes). Figure 1 illustrates the VDJ recombination process that selects one V gene, one D gene, and one J gene and concatenates them, thus generating an *immunoglobulin gene* that encodes an antibody. In our discussion, we hide some details to make the paper accessible to information theorists without immunology background. For example, although there are multiple immunoglobulin loci in the human genome, we limit attention to the 1.25 million-nucleotide long *immunoglobulin heavy chain locus*. Although we stated above that an immunoglobulin gene encodes an antibody, in reality it encodes only the *heavy chain* of an antibody (antibodies are formed by both heavy and light chains).

Since the described process can generate only 40 × 27 × 6 = 6480 antibodies, it cannot explain the astonishing diversity of human antibodies. However, the VDJ recombination is more complex than this: it deletes some nucleotides at the start and/or the end of V, D, J genes and inserts short stretches of randomly generated nucleotides (*non-genomic insertions*) between V-D and D-J junctions. Such *insertions and deletions* (*indels*) greatly increase the diversity of antibodies generated through the VDJ recombination process. But this is only the beginning of the molecular process that further diversifies the set of antibodies.

Somatic hypermutations and clonal selection: Indels greatly increase the diversity of antibodies but even this diversity is insufficient for neutralizing a myriad of antigens that the organism might face. However, the VDJ recombination generates sufficient diversity to achieve an important goal—some of the generated antibodies in this huge collection bind to a specific antigen, albeit with low *affinity* (i.e., the strength of antibody-antigen binding) that is insufficient for neutralizing the antigen. The adaptive immune system uses an ingenious evolutionary mechanism for gradually increasing the affinity of binding antibodies and thus eventually neutralizing an antigen [34].

Once an antibody binds to an antigen (even an antibody with a low affinity), the corresponding immunoglobulin gene undergoes random mutations (referred to as *somatic hypermutations* or *SHMs*) that can both increase and reduce the affinity of an antibody. To enrich the pool of antibodies with high affinity, these mutations are iteratively accompanied by the *clonal selection* process that eliminates antibodies with low affinity (Figure 1). The iterative somatic hypermutations and clonal selection are not unlike an extremely fast evolutionary process that generates a huge variety of antibodies from a single initial antibody and eventually leads to generating a new high-affinity antibody able to neutralize an antigen.

Personalized immunogenomics: Modern DNA sequencing technologies sample the set of antibodies by generating sequences of millions of randomly selected immunoglobulin genes (*antibody repertoire*) out of ≈ 10^8 distinct antibodies circulating in our body. Analysis of antibody repertoires across various patients opens new horizons for developing antibody-based drugs, designing vaccines, and finding associations between genomic variations in the immunoglobulin loci and diseases. The emergence of antibody repertoire datasets in the last decade raised new algorithmic problems that remain largely unsolved.

The immunoglobulin locus is a highly variable region of the human genome—the sets of V, D, and J genes (referred to as *germline genes*) differ from individual to individual. Identifying germline V, D, and J genes in an individual is important since variations in these genes have been linked to various diseases [35], differential response to infection, vaccination, and drugs [36], and disease susceptibility [35], [37]. The ImMunoGeneTics (IMGT) database of variations in germline genes remains incomplete even in the case of well-studied human genes [38]. In the case of immunologically important model organisms, such as camels or sharks, the germline genes remain largely unknown. Unfortunately, since assembling the sequence of the highly repetitive immunoglobulin locus faces challenges [39] and does not provide one with information on how various germline genes contribute to an antibody repertoire, the efforts like the 1,000 Genomes Project have resulted only in limited progress toward inferring the population-wide census of human germline genes [40], [41].

Since the information about germline genes in an individual (personalized immunogenomics data) is typically unavailable, researchers use the reference genes instead of personal germline genes, thus limiting various immunogenomics applications. Personalized immunogenomics studies attempt to derive the germline genes by analyzing antibody repertoires. Each antibody can be viewed as a trace generated from the three sets of unknown seed strings (all V genes, all D genes, and all J genes in an individual) through the VDJ recombination and somatic hypermutations (Figure 1). Hence, one can reformulate reconstruction of germline genes from an antibody repertoire as a novel Trace Reconstruction Problem. In Section III, we describe a series of problems with gradually increasing complexity that model antibody generation from the germline genes.

B. Trace Reconstuction in DNA Data Storage

DNA has emerged as a potentially viable storage medium for large quantities of digital data [24]-[33]. A digital file can be encoded by a collection of DNA sequences where each individual sequence represents a small part of the data. One application is archival storage, where DNA promises to have orders of magnitude improved data density and durability as compared to existing storage media (e.g., magnetic tapes or solid state). The field is rapidly growing, and current DNA data storage systems can store and retrieve hundreds of megabytes of data, with many additional features such as random data access [30]. We provide an overview of DNA data storage and highlight the role that trace reconstruction plays in the data retrieval process [30], [33]. Figure 2 depicts the core components of the storage and retrieval pipeline.

Storing the data: Storing a file in DNA involves several steps. First, the digital file is compressed and partitioned into small, non-overlapping blocks. Then, each individual block is either encoded using an error-correcting code or is randomized using an independent pseudo-random sequence. This provides a set of strings that encode the content of the digital file. To store the location of each block, an address is added to each string. Finally, a global error-correcting code is applied to the resulting set of strings, and the strings are translated into the $\{A, C, G, T\}$ alphabet. If multiple files are stored together, then a file identifier is also added in the form of a *DNA primer* (a short nucleotide string). This process results in a large collection of short strings (for example, millions of strings, each containing hundreds of characters). This set of strings, which we call *seed strings*, are then synthesized into real DNA molecules and stored in a tube until the file is ready to be retrieved. The synthesis process generates many copies of each seed string.

Retrieving the data: The stored information is read using standard DNA sequencing machines such as DNA sequencers produced by Illumina. A small amount of DNA is extracted from a tube so that the remaining DNA may be used for other retrieval attempts later on. Since this amount may be insufficient for reading DNA (sequencing machines have limitations with respect to the minimal amount of DNA they can sequence), the extracted DNA is amplified using *polymerase chain reaction* (*PCR*) to generate multiple copies (e.g., 5–30) of each DNA molecule in a sample. The PCR step enables random access retrieval—to access a subset of files, it suffices to copy and sequence the subset of seed strings with *primers* (file IDs) corresponding to these files.

However, the PCR process introduces additional errors in each of the amplified copies. Since DNA sequencing machines are not able to identify the error-free sequence of nucleotides in a DNA molecule, they add extra errors to the previously introduced amplification errors. The combination of amplification and sequencing errors typically results in ≈ 1–2% error rate (substitutions and indels). However, there is some debate about the rate and the most common type of error [42], [43]. The output of sequencing is a set of strings that contains several error-prone copies (called *reads*) of each originally synthesized seed string. Much longer seed strings (tens of thousands of nucleotides versus hundreds of nucleotides in existing applications) can be sequenced using the recently emerged *long-read sequencing technologies* but the current error rate of such technologies is ≈ 10%, with a large proportion of indels [30], [33], [44], [45].

DNA data retrieval as a trace reconstruction problem: After the sequencing reads are generated, the goal is to recover the seed strings from the observed error-prone reads that have indels and substitutions. The first challenge is to determine which reads correspond to which seed strings by clustering reads so that each cluster contains the error-prone copies derived from a single seed string [46]. In some DNA data storage systems, the seed strings are randomized or encoded in a such a way that they have large pairwise edit distance [30], [47]-[49]. This property simplifies the clustering problem because the underlying clusters are well-separated. In this context, clustering algorithms have been developed that scale to billions of reads [46].

Recovering the seed strings from the reads can be formulated as a trace reconstruction problem. Each seed string is observed a small number of times, where the error-prone copies (traces) correspond to the reads in a cluster. The objective is to recover as many seed strings as possible. A small number of missing or erroneous seed strings may be tolerated because of the error correction methods. Consequently, it suffices to ensure that a reconstruction algorithm recovers a seed string with probability *ReconstructionRate*, where the exact success probability depends on the amount of redundancy in the error-correcting code (e.g., the default value may be *ReconstructionRate* = 0.95). There is a trade-off where having more traces leads to lower error rate in reconstruction, but it incurs a higher sequencing cost and time. In practice, it is typical to use clusters that contain 5–30 reads (traces) [30].

While we focus on trace reconstruction problems in DNA data storage, there are many other challenges and recent results, including better automation methods [50], [51], alternative synthesis schemes [52], [53], improved density and robustness using codes [27], [44],

[54]-[66], and more realistic error models and fundamental limits [16], [67]-[70]. For more details about DNA data storage, see the following surveys and references therein [24], [32].

C. Similarities and differences of the two applications

The trace reconstruction problems for immunogenomics and DNA data storage are distinct, both in terms of the trace generation models and how well the models have been studied in the literature.

In immunogenomics, the traces contain important mutations that are introduced during the VDJ recombination and somatic hypermutagenesis. While sequencing and amplification technologies also introduce errors in sequenced antibody repertoires, their rate is much lower compared to the mutations introduced at the antibody generation step. Therefore, we ignore sequencing and amplification errors in immunogenomics applications and focus on the mutations. In contrast, the errors in the DNA data storage applications are only because of the artifacts of the process used to access the data stored in DNA and thus cannot be ignored.

The seed strings in immunogenomics represent real genetic data, whereas the DNA data storage sequences are synthesized to represent information in a digital file. While there has been a considerable amount of work in trace reconstruction problems motivated by DNA data storage, trace reconstruction studies in immunogenomics have only started to emerge [22], [23].

D. Outline

The rest of the paper is organized as follows. In Section II, we introduce the algorithmic and information-theoretic formulations of trace reconstruction. Section III describes trace generation in computational immunogenomics. In Sections III-A and III-B, we introduce the D genes trace reconstruction problem. In Section III-C, we introduce a more complex problem of reconstructing V, D, and J genes that are concatenated together to form antibodies. Section IV describes the theoretical formulation of trace reconstruction problems for DNA data storage. In Section V, we survey theoretical results and practical solutions to the trace reconstruction problem for the deletion channel, along with open problems relevant to developing DNA data storage. Finally, in Section VI we propose several directions for future work.

II. Algorithmic and information-theoretic formulations

In this section, we formalize the algorithmic goals of the trace reconstruction problems. We begin by considering an abstract model, where a single, unknown seed string s generates a random trace c with probability $\mathbf{Pr}(c \mid s)$. For each possible trace c and seed string s, the model specifies $\mathbf{Pr}(c \mid s)$. To recover the seed string s, the reconstruction algorithm receives a collection of traces generated from s, which we refer to as a *trace-set* $C = \{c_1, c_2, \ldots, c_T\}$. For simplicity, we assume that the traces are independent and identically distributed, and hence,

$$\mathbf{Pr}(C \mid s) = \prod_{i=1}^{T} \mathbf{Pr}(c_i \mid s).$$

Given an integer T, we use $\mathscr{C}_T$ to denote the collection of all possible sets of T strings over a fixed alphabet, and we note that $\sum_{C \in \mathscr{C}_T} \mathbf{Pr}(C \mid s) = 1$.

We also consider cases where the generation process involves sets of seed strings. In these cases, one string is sampled at a time from a set, and the traces are independently generated from the sampled strings (sometimes concatenating groups of traces to obtain the final trace-set). For example, we can consider the two step process where a seed string s is uniformly randomly selected from an unknown *seed-set* S, and then s generates a trace. Given a seed-set $S = \{s_1, \ldots, s_M\}$, the probability of a trace-set C is

$$\mathbf{Pr}(C \mid S) = \prod_{i=1}^{T} \mathbf{Pr}(c_i \mid S) = \prod_{i=1}^{T} \left(\frac{1}{M} \sum_{j=1}^{M} \mathbf{Pr}(c_i \mid s_j) \right).$$

In other words, in the multiple seed string case, we can still define the probability of a trace-set C in terms of the probability of generating a single trace from a single seed string. The goal is to recover all or most of the strings in S by using a trace-set generated via this two step process.

We next discuss how to evaluate a reconstruction algorithm A that takes as input the trace-set C and outputs a string $A(C)$. We assume that the algorithm knows the trace generation model, that is, for any trace c and seed string s, it knows the probability $\mathbf{Pr}(c \mid s)$. The goal is to reconstruct the seed string using the traces. The fact that the traces themselves are random means that there are at least two ways to evaluate a reconstruction algorithm.

The *maximum likelihood estimate (MLE)* is a string $\hat{s}$ that maximizes $\mathbf{Pr}(C \mid \hat{s})$ among all seeds. As the probabilities are known to the algorithm, the MLE can always be computed by exploring all strings (i.e., brute-force search) as long as the set of possible candidate strings is finite. The trace generation models that we consider have the property that the maximum length of the seed string can be inferred from the trace-set with high probability. Therefore, the brute-force search can be taken over a finite set of strings. For some models, an efficient algorithm computing the MLE is known, with running time that is polynomial in the number T of traces and the length $|s|$ of the seed string (see Section III-A). However, for many trace generation models, computing the MLE in polynomial time is currently an open question (i.e., the only known solution is brute-force search).

To circumvent the difficulty of the maximum likelihood objective, previous work instead measures the probability that an algorithm outputs the seed string s used to generate the traces. The trace-set is viewed as a random input, and the probability is taken over the randomness in the trace generation process. We start with definitions for a fixed, but unknown, seed string s, and we later also consider s itself being random. Define the *success probability* of an algorithm A and a seed string s as

$$P_A(s, T) = \sum_{C \in \mathscr{C}_T} \mathbf{Pr}(C \mid s) \cdot 1_{\{A(C) = s\}},$$

where $1_{\{A(C)=s\}}$ is the indicator function for the event $\{A(C) = s\}$ that the algorithm outputs the seed string s. It is straightforward to extend the definition of $P_A(s, T)$ to randomized algorithms; the output $A(C)$ would also be a random variable, and the term $1_{\{A(C)=s\}}$ would be replaced with $\mathbf{Pr}(A(C) = s)$.

Let $\mathscr{U}$ be a universe of possible seed strings (e.g., all strings of a certain length over a binary or quaternary alphabet). We define the *worst-case success probability* of algorithm A for trace-sets of size T over universe $\mathscr{U}$ as

$$P_A(\mathscr{U}, T) = \min_{s \in \mathscr{U}} P_A(s, T).$$

Then, the *worst-case* trace reconstruction problem is to develop an efficient algorithm that maximizes $P_A(\mathscr{U}, T)$. The definition above guarantees that the algorithm succeeds with probability at least $P_A(\mathscr{U}, T)$ when s is an arbitrary seed string from the universe and the trace-set has size T.

We also consider the *average-case* trace reconstruction problem, where the seed string s is chosen uniformly at random from the universe (instead of being arbitrary, as in the worst-case version). More precisely, the goal is to develop an efficient algorithm A that maximizes the *average-case success probability*, which is defined as

$$\widetilde{P}_A(\mathscr{U}, T) = \frac{1}{|\mathscr{U}|} \sum_{s \in \mathscr{U}} P_A(s, T).$$

Notice that the probability here is taken over both the seed string s and the trace-set C. The average-case formulation leads to a nice connection to the MLE. Expanding $P_A(s, T)$, we have that

$$\begin{aligned} \widetilde{P}_A(\mathscr{U}, T) &= \frac{1}{|\mathscr{U}|} \sum_{s \in \mathscr{U}} \sum_{C \in \mathscr{C}_T} \mathbf{Pr}(C \mid s) \cdot 1_{\{A(C) = s\}} \\ &= \sum_{C \in \mathscr{C}_T} \frac{1}{|\mathscr{U}|} \sum_{s \in \mathscr{U}} \mathbf{Pr}(C \mid s) \cdot 1_{\{A(C) = s\}} \end{aligned}$$

Therefore, the inner sum over $s \in \mathscr{U}$ is maximized when A outputs the string $\hat{s}$ maximizing $\mathbf{Pr}(C \mid \hat{s})$, or in other words, when the algorithm outputs the MLE.

We note that algorithm does not know the seed string, and hence, it cannot determine whether it outputs s or some other string s' that could have generated the trace-set. In contrast, the MLE is always rigorously defined because it allows the algorithm to output any string that maximizes the likelihood. To rigorously reason about the maximum success probability formulation, we assume that the trace-set is large enough so that a unique seed string must have generated the traces with high probability, and hence, the algorithm can recover this string with high probability. Later on, we also discuss how to empirically determine the success probability with a benchmark dataset.

In summary, when the seed string is random (i.e., the average-case version), then the maximum likelihood solution also maximizes the success probability $\widetilde{P}_A(\mathcal{U}, T)$. In particular, the ideal solution to the average-case trace reconstruction problem would be an efficient algorithm that computes the MLE, with running time that is polynomial in the number of traces and the seed string length. For the new models that we introduce, we remain hopeful that such an algorithm can be found. However, the only presently known algorithm for all but one of the models is to perform brute-force search. Moreover, in the worst-case version, the MLE may not maximize the success probability $P_A(s, T)$, and these two formulations may lead to different optimal algorithms.

In Section III, we introduce various trace generation models in computational immunogenomics. For each model, we provide a problem statement that asks for an MLE solution, i.e., an algorithm that outputs a seed string (or a seed-set) that maximizes the likelihood of a given trace-set. However, we also note that it would be valuable to develop an algorithm with high success probability when the input is viewed as a random trace-set. While both of these are valid and important formulations, the MLE version is a long-standing tradition in bioinformatics that is widely used in such areas as computing phylogenetic trees [71] and genetic linkage analysis [72]. In immunogenomics, MLE was used for computing antibody clonal trees [73], modeling VDJ recombination [74], [75], and modeling antibody-antigen interactions [76], [77]. On the other hand, information theory and computer science researchers may prefer to develop (approximation) algorithms that are evaluated based on their success probability. Therefore, we briefly discuss evaluation metrics before introducing the models.

A. Approximation Algorithms and Empirical Success Probability

As for many other bioinformatics problems, since brute-force solutions are prohibitively slow, the goal is to develop fast approximation or heuristic algorithms that are practical for typical input sizes. For an analogy, although the edit distance problem between two sequences can be solved in polynomial time [78], the closely related *sequence alignment* problem between multiple sequences is NP-hard [79]. Nevertheless, since the multiple sequence alignment problem is at the heart of sequence comparison in bioinformatics, hundreds of heuristic algorithms have been developed for solving it [80]. The ultimate goal of these algorithms is to generate biological insights, and hence, they are often benchmarked on datasets with known solutions [81].

Turning back to trace reconstruction problems, it would often suffice to output the MLE on most trace-sets, instead of all of them (e.g., failing with vanishingly small probability). Alternatively, when it is difficult to find the entire seed string s maximizing $\mathbf{Pr}(C \mid s)$, it may be possible to find a sufficiently long substring instead. Doing so could further enable finding the entire seed string through a complementary experimental approach. For example, a seed string reconstructed by an approximation or heuristic algorithm can be later validated and error-corrected by using genomics data that complements the immunogenomics data [23].

We also mention one more choice: is the number of traces fixed in advance or not? For a fixed number of traces T, the goal is to design an algorithm with highest possible success probability. Alternatively, since the success probability increases as T increases, we consider an additional input parameter *ReconstructionRate*, where $0 \leq ReconstructionRate \leq 1$ and the goal is to design an algorithm with success probability surpassing *ReconstructionRate* using as few traces as possible. Formally, we want to determine the minimum value T^* such that the trace reconstruction problem with T traces is feasible for a given *ReconstructionRate* as long as $T \geq T^*$. This value T^* is called the trace complexity, and we discuss it further in Section IV. We also note that the success probability can be driven to one by using more traces, assuming it starts above 0.5. Indeed, taking the majority vote over $O(\log(1/\beta))$ trials for any value $0 < \beta < 1$ will lead to success probability $1 - \beta$, which follows via a Chernoff bound. Both algorithmic formulations are relevant for practical applications.

For the immunology models, we consider a fixed number of traces. The reason is that the number of traces depends on multiple factors—such as the reconstruction of *clonal trees* during antibody development [82] or selecting the best candidate for follow-up antibody engineering efforts [83]—and accurate reconstruction of germline genes is only one of them. For the DNA data storage models, we consider an information-theoretical perspective and focus on determining the minimum number of traces that suffice for a certain success probability.

The average-case success probability can be empirically calculated by choosing the seed string s at random and testing whether $A(C) = s$ when the trace-set is generated at random from s. For the worst-case success probability, it is infeasible to compute the minimum over all possible length n strings. Instead, it would be easier to use seed strings from a benchmark dataset. For example, if the *ReconstructionRate* is 0.95, then the algorithm will likely output $A(C) = s$ at least 95 times over 100 randomly generated trace-sets, and this should hold for each seed string s from the dataset. In the DNA data storage application, the seed strings are constructed synthetically during the storage process, and therefore, they may be used as a benchmark.

III. Trace Generation in Computational Immunogenomics

Reconstructing D genes is more difficult than reconstructing V and J genes:

Inferring the sequences of germline genes using immunosequencing data obtained from an individual antibody repertoire is an important problem [22], [23], [84]-[87]. In the case of V and J genes, this challenge was addressed by [85]-[88]. Reconstruction of shorter D genes is a more challenging task [88]. D genes contribute to the *complementarity determining region 3* (*CDR3*) that spans the V-D and D-J junctions and represents an important and highly divergent part of antibodies that accumulates many SHMs. Since D genes typically get truncated on both sides during VDJ recombination, the CDR3 typically contains a truncated D gene. Each CDR3 also contains some random insertions at the V-D and D-J junctions. These truncations and insertions, combined with the fact that D genes are much shorter than

V and J genes, make the task of aligning various CDR3s (and thus aligning segments of D genes that survive within these CDR3s) more difficult than alignment of longer and typically less mutated fragments of immunoglobulins that originated from V and J genes.

The biologically adequate problem formulations in immunogenomics are rather complex, making it difficult to develop and test algorithmic ideas for solving these problems. That is why the usual path toward solving such problems is to start from simple and often inadequate formulations that however shed light on algorithmic ideas that can be used for solving more complex problems [89]. We follow this path by starting with a simple formulation for the problem of inferring D genes from CDR3s extracted from an antibody repertoire. Although efficient algorithms for the complex biologically adequate problems remain unknown, the recently developed MINING-D heuristic [23] led to the discovery of previously unknown D genes across multiple species. After describing open problems relevant to finding new D genes, we formulate more difficult problems relevant to inferring the sets of V, D, and J genes (rather than D genes only).

Generating CDR3 from a D gene:

We denote the length of a string s as $|s|$ and the concatenation of strings s_1 and s_2 as $s_1 * s_2$. We refer to a random string of length l (each symbol is generated uniformly at random from a fixed alphabet $\mathscr{A}$) as r^l. Given an integer t, we define a random string $Random_{\leq t}$ as r^l, where an integer l is sampled uniformly at random from [0, t]. In this paper, $\mathscr{A} = \{A, G, C, T\}$.

Below we describe various models for generating traces from a seed string or from a seed-set. In all models, we assume that each trace is generated independently. To model generation of a CDR3 (trace) from a D gene (seed) in the models below, we describe the following operations on a string s (Figure 3):

- *Trim(s)*: A pair of integers l and k are sampled uniformly at random from the set of all pairs of non-negative integers (i, j) satisfying the condition $i + j \leq |s|$. The prefix of length l and the suffix of length k of s are trimmed.
- *Mutate$_\varepsilon$(s)*: Each letter in s is independently mutated with probability ε in such a way that mutations into all $|\mathscr{A}| - 1$ symbols (differing from the symbol in s) are equally likely.
- *Extend$_t$(s)*: a string $R_1 * s * R_2$ where R_1 and R_2 are independent instances of $Random_{\leq t}$.

Figure 3 illustrates the *Extend$_t$(Mutate$_\varepsilon$(Trim))* model for generating a CDR3 from a D gene using random deletions/insertions and somatic hypermutations. Before considering this rather complex model, we will consider a series of simpler (albeit less adequate) models for generating CDR3s (Figure 4) that use the operations listed below.

- *TrimSuffix(s)*: an integer k is sampled uniformly at random from [0, $|s|$] and the suffix of s of length k is trimmed.

- *TrimSuffixAndExtend(s)*: an integer k is sampled uniformly at random from [0, $|s|$], the suffix of s of length k is trimmed, and the resulting string is concatenated with r^k.
- *SuffixExtend$_t$(s)*: a string $s * Random_{\leq t}$.
- *TrimAndExtend(s)*: a pair of integers l and k are sampled uniformly at random from the set of all pairs of non-negative integers (i, j) satisfying the condition $i+j \leq |s|$. The prefix of length l and the suffix of length k of s are trimmed resulting in a string *Trim(s)*. *TrimAndExtend(s)* is defined as $r^l * Trim(s) * r^k$.

We will start with a simple *TrimSuffixAndExtend* model where the seed string and the modified strings are of equal lengths. The next *SuffixExtend$_t$(TrimSuffix)* model relaxes the assumption that the lengths of all modified strings generated from a seed string are the same since the same D gene can produce CDR3s of different lengths in the VDJ recombination process. In the next *SuffixExtend$_t$(Mutate$_\varepsilon$(TrimSuffix))* model, we further allow mutations to occur in the seed string. This is important because the immune system introduces random somatic hypermutations to increase the affinity towards an antigen.

In the above models, only the suffix of the seed string gets trimmed in the first step. In the real VDJ recombination process, however, D genes get trimmed on both sides. To incorporate this fact in the above models, we next present the *TrimAndExtend* model that allows trimming on both sides while keeping the lengths of the modified strings the same. This is analogous to the *TrimSuffixAndExtend* model and the only difference between the two models is that the former gets trimmed on both sides whereas in the latter, only the suffix is trimmed. To introduce mutations in this model, where the seed string gets trimmed on both sides, we then present the *Mutate$_\varepsilon$(TrimAndExtend))* model, while still keeping the lengths of all modified strings the same. Finally, to allow for the possibility of different lengths of modified strings, while keeping intact the trimming from both sides and the random mutations, we introduced the *Extend$_t$(Mutate$_\varepsilon$(Trim))* model which is the most biologically adequate model for VDJ recombination among all introduced models.

All models presented in the next subsection can be extended to the multiple seed strings case where a seed string is chosen randomly from a seed-set, a trace is then generated from the chosen seed string according to a model, and the process is independently repeated a number of times to generate a set of traces. In Section V-A, we will discuss the *population recovery problem*, which also concerns reconstructing multiple seed strings under a different trace generation model.

The average length and the number of D genes varies among species—for humans and many immunologically important mammals (e.g., mice and rats), the length of D genes does not exceed 40 nucleotides and the number of D genes varies from 20 to 40. In contrast, other immunologically important mammals (e.g., cows) have long (150 nucleotides) and very repetitive D genes. Since future personalized immunogenomics studies may involve thousands or even millions of individuals, the D gene reconstruction algorithms must scale accordingly, e.g., the running time should not exceed a few hours.

A. A Simple but biologically inadequate model for D gene reconstruction

TrimSuffixAndExtend: Although this model (Figure 4a) does not adequately reflect the realities of VDJ recombination, the trace reconstruction problem for this model can be efficiently solved. A seed string may generate the same trace for different values of the trimming integer k in the *TrimSuffixAndExtend* model. The probability $\mathbf{Pr}(c \mid s)$ that a seed string s generates a trace c depends only on the length m of their longest shared prefix and is given by

$$\begin{aligned}\mathbf{Pr}(c \mid s) &= \frac{1}{|s| + 1}\sum_{k=0}^{m}\frac{1}{|\mathscr{A}|^{|s| - k}}\\ &= \frac{1}{(|s| + 1)(|\mathscr{A}|^{|s|})} \times \frac{|\mathscr{A}|^{m+1} - 1}{|\mathscr{A}| - 1}\\ &= K(|s|, |\mathscr{A}|) \times (|\mathscr{A}|^{m+1} - 1)\end{aligned}$$

where $K(|s|, |\mathscr{A}|)$ is constant given the length of the seed string and the alphabet size. The probability that a seed string s generates a trace-set $C = \{c_1, c_2, \ldots, c_T\}$ is computed as

$$\mathbf{Pr}(C \mid s) = \prod_{i=1}^{T}\mathbf{Pr}(c_i \mid s)\,. \tag{1}$$

Trace Reconstruction Problem in the *TrimSuffixAndExtend* model

Input: A trace-set C generated from an unknown seed string according to the *TrimSuffixAndExtend* model.

Output: A string s maximizing $\mathbf{Pr}(C \mid s)$.

Solving Trace Reconstruction Problem in the TrimSuffixAndExtend model: $\mathbf{Pr}(C \mid s)$ is maximized by one of the traces. This observation leads to an algorithm for solving the String Reconstruction Problem (with complexity $O(|s| \cdot T^2)$) that simply computes $\mathbf{Pr}(C \mid s)$ for each of the T traces. We describe an improved algorithm for solving this problem with a running time of $O(|s| \cdot T)$, which is linear in the input size.

Maximizing $\mathbf{Pr}(C \mid s)$ is equivalent to maximizing $\prod_{i=1}^{T} K(|s|, |\mathscr{A}|) \times (|\mathscr{A}|^{m_i + 1} - 1)$, where m_i is the length of the longest shared prefix between s and c_i [23]. Since $K(|s|, |\mathscr{A}|)$ is a constant, it is equivalent to finding a string s that maximizes

$$\text{score}(C \mid s) = \sum_{i=1}^{T}\log(|\mathscr{A}|^{m_i + 1} - 1)\,.$$

We denote $f(j) = \log(|\mathscr{A}|^{j+1} - 1)$ and search for a string s that maximizes $\sum_{i=1}^{T} f(m_i)$ where m_i is the length of the longest shared prefix between s and c_i. We denote a t-symbol prefix (*t-prefix*) of a string c as c^t and the set of all t-prefixes of strings from C as C^t. Given a string s and an integer t, we say that a string c is *t-similar* to s if t-prefixes of s and c

coincide. The number of strings in C that are t-similar to s is denoted as $sim_t(C, s)$. Given a string s,

$$\begin{aligned} \mathrm{score}(C^t \mid s^t) = {} & \mathrm{score}(C^{t-1} \mid s^{t-1}) \\ & + sim_t(C, s) \times \log\left(\frac{|\mathscr{A}|^{t+1} - 1}{|\mathscr{A}|^{t} - 1}\right). \end{aligned} \tag{2}$$

We use this recurrence to efficiently compute score $(C \mid s)$ for each string s from C using dynamic programming on a trie constructed from all traces in C [90]. Each vertex in the trie is a t-prefix s^t of a string from C, and we recursively compute score$(C^t \mid s^t)$ in each vertex of the trie using the above recurrence assuming that the score of the root is $\log(|\mathscr{A}| - 1)$. The optimal string corresponds to the leaf node with the maximum score.

For all strings in C and all values of t, the quantities $sim_t(C, s)$ can be computed during the construction of the trie as follows. Traces are added sequentially to construct the trie. In addition to t-prefixes, each vertex also stores $sim_t(C, s)$ which is initialized to 1 for a new vertex. For example, in Figure 5, we start with an empty trie and first add the trace “CATTAT” by creating six new vertices, each representing one of the six t-prefixes. At this point, the trie contains only one string, and for all vertices, we have $sim_t(C, s) = 1$. Then, we add the next trace “CATTTG”. For $t \leq 4$, the t-prefixes of “CATTAT” and “CATTTG” coincide. In other words, they share the first four vertices in the trie. For all vertices that are traversed while inserting a new trace, the values of $sim_t(C, s)$ are updated by adding 1 to the current values. For new vertices, like before, the values of $sim_t(C, s)$ are initialized to 1. In this example, for the vertices representing t-prefixes “C”, “CA”, “CAT”, and “CATT”, the value of $sim_t(C, s)$ will be updated to 2, whereas for the two new vertices representing t-prefixes “CATTT” and “CATTTG”, the values of $sim_t(C, s)$ will be 1. All traces are inserted to the trie in this manner. We can thus compute all $sim_t(C, s)$ values during the construction of the trie with complexity $O(|s| \cdot T)$. After the construction of the trie, all quantities score$(C^t \mid s^t)$ can then be computed by a single Depth-First Search using Eq. (2).

TrimSuffixAndExtend model with multiple seeds: Next, we consider a modified *TrimSuffixAndExtend* model with a seed-set $S = \{s_1, s_2, \ldots, s_M\}$. Traces are generated via a two step approach. First, a string $s_i \in S$ is chosen uniformly randomly from S. Then, s_i is modified to generate a trace c according to the *TrimSuffixAndExtend* model. We note that S can either be an arbitrary set of M strings (worst-case) or the strings in S can be chosen independently and uniformly from the universe of possible strings (average-case). Note that the above model is described for a uniform distribution over the seed strings. In the real VDJ recombination process, various D genes contribute to immunoglobulin genes with varying propensities. To incorporate this fact, the above model can be reformulated by considering an arbitrary distribution on the seed strings.

> Trace Reconstruction with Multiple Seeds Problem in the *TrimSuffixAndExtend* model
>
> **Input:** A trace-set C generated from an unknown set of M seed strings of the same length according to the *TrimSuffixAndExtend* model.

Output: A set of strings $S = \{s_1, s_2, \ldots, s_M\}$ maximizing $\mathbf{Pr}(C \mid S)$.

The MINING-D heuristic algorithm: Although the trace reconstruction problem can be efficiently solved in the *TrimSuffixAndExtend* model, it is unclear how to generalize the algorithm for the more complex models with multiple D genes and varying lengths of modified strings. Bhardwaj et. al. [23] propose a practical greedy heuristic for this model that, while being suboptimal, motivates practical algorithms for more complex models.

For the *TrimSuffixAndExtend* model, the algorithm starts with an empty string and at step j extends it on the right by the most abundant symbol in C at position j and discards from C the strings that have symbols that are not the most abundant symbols at position j. This procedure repeats until the length of the resulting string equals the length of the seed string s. This greedy algorithm, however, cannot be directly used in practice because (a) the CDR3s are formed by multiple D genes, (b) the number of D genes is unknown *a priori*, (c) the D genes have different lengths that are unknown, (d) CDR3s generated by the same D gene can have different lengths.

The MINING-D algorithm [23], inspired by the above greedy algorithm, considers the complexities of the real immunogenomics data. It uses the observation that, although D genes typically get truncated on both sides during the VDJ recombination process, their truncated substrings are often present in the newly recombined genes, and, hence, the CDR3s. Therefore, we expect the truncated substrings of D genes to be highly abundant in a CDR3 dataset. MINING-D starts by finding the most abundant k-mers (a k-mer is a string of length k). It then extends them on both sides using the greedy algorithm to recover entire D genes that contain highly abundant k-mers as substrings. MINING-D defines a probabilistic stopping rule as the lengths of the D genes are not known *a priori*. This stopping rule also allows us to recover D genes of different lengths. Since some abundant k-mers can be substrings of multiple D genes, MINING-D allows multiple extensions from each k-mer in the extension procedure.

We next introduce models that incorporate more complexities of the VDJ recombination process, leading up to the model that mimics the real formation of an immunoglobulin gene from a set of V, D, and J genes. To the best of our knowledge, these models have not been studied in the literature and brute-force search is the only known exact solution to trace reconstruction in these models.

B. Toward a biologically adequate model for D gene reconstruction

SuffixExtend$_t$(TrimSuffix): Unlike the *TrimSuffixAndExtend* model, the *SuffixExtend$_t$(TrimSuffix(s))* model (Figure 4b) generates traces of varying lengths from a single seed string s. Let s_{trim} be the substring of s that remains after the operation *TrimSuffix* is applied on s. Then, $\mathbf{Pr}(c \mid s)$ is given by

$$
\begin{aligned}
\mathbf{Pr}(c \mid s) &= \sum_{k=0}^{|s|} \Pr(c, |s_{trim}| = k \mid s) \\
&= \sum_{k=0}^{|s|} \Pr(|s_{trim}| = k \mid s) \Pr(c \mid s, |s_{trim}| = k) \\
&= \frac{1}{(|s| + 1)} \sum_{k=0}^{|s|} \Pr(c \mid s, |s_{trim}| = k)
\end{aligned}
$$

Let m be the length of the longest shared prefix between c and s, as before. Then, $\Pr(c \mid s, |s_{trim}| = k)$ is non-zero only if $|c| - t \leq k \leq m$ and can be written as

$$
\Pr(c \mid s, |s_{trim}| = k) = \begin{cases} \dfrac{1}{(t+1)|\mathscr{A}|^{|c| - k}} & \text{if } |c| - t \leq k \leq m \\ 0 & \text{otherwise} \end{cases}
$$

Thus $\mathbf{Pr}(c \mid s)$ is zero if $m < |c| - t$. Otherwise,

$$
\mathbf{Pr}(c \mid s) = \frac{1}{(|s| + 1)(t + 1)} \sum_{k = (|c| - t)^{+}}^{m} \frac{1}{|\mathscr{A}|^{|c| - k}} \tag{3}
$$

where $x^{+} = max(x, 0)$.

- Trace Reconstruction Problem in the *SuffixExtend$_t$(TrimSuffix(s))* model
- **Input:** A trace-set C generated from an unknown seed string according to the *SuffixExtend$_t$(TrimSuffix(s))* model.
- **Output:** A string maximizing $\mathbf{Pr}(C \mid s)$.

SuffixExtend$_t$(Mutate$_\varepsilon$(TrimSuffix)): We now consider a slightly more realistic model for trace generation that incorporates somatic hypermutations (Figure 4c). The probability $\mathbf{Pr}(c \mid s)$ that a seed string s generates a trace c is given by

$$
\mathbf{Pr}(c \mid s) = \frac{1}{(|s| + 1)(t + 1)} \times \sum_{k = (|c| - t)^{+}}^{|s|} \frac{(1 - \varepsilon)^{k - d_k} (\varepsilon / (|\mathscr{A}| - 1))^{d_k}}{|\mathscr{A}|^{|c| - k}}
$$

where d_k is the Hamming distance between the prefixes of c and s of length k.

- Trace Reconstruction Problem in the *SuffixExtend$_t$(Mutate$_\varepsilon$(TrimSuffix))* model
- **Input:** A trace-set C generated from an unknown seed string according to the *SuffixExtend$_t$(Mutate$_\varepsilon$(TrimSuffix))* model.
- **Output:** A string maximizing $\mathbf{Pr}(C \mid s)$.

TrimAndExtend: In all the models above, only the suffix of the seed string gets trimmed in the first step. In contrast, during the VDJ recombination process, the D gene gets trimmed from both sides. We will thus consider the *TrimAndExtend* model (Figure 4d) for generating a trace c from a seed string s.

Since strings s and c have the same length, their comparison results in a binary comparison vector where 1s (0s) correspond to the match (mismatch) positions. Let $t(i)$ denote the length of the continuous run of 1s starting at position $i + 1$ in the comparison vector. The probability that a seed string s generates a trace c is given by

$$\mathbf{Pr}(c \mid s) = \frac{2}{(\mid s \mid +1)(\mid s \mid +2)} \sum_{i=0}^{\mid s \mid} \left(\sum_{k = \mid s \mid - i - t(i)}^{\mid s \mid - i} \frac{1}{\mid \mathscr{A} \mid^{i+k}} \right)$$

- Trace Reconstruction Problem in the *TrimAndExtend* model
- **Input:** A trace-set C generated from an unknown seed string according to the *TrimAndExtend* model.
- **Output:** A string maximizing $\mathbf{Pr}(C \mid s)$.

Mutate$_\varepsilon$(TrimAndExtend): We now consider a model that incorporates mutations in the *TrimAndExtend* model (Figure 4e). Let substring$_{l,k}(s)$ be the substring of the seed string s where the prefix of length l and the suffix of length k have been trimmed. The probability that a seed string s generates a trace c in the *Mutate$_\varepsilon$(TrimAndExtend)* model is given by

$$\mathbf{Pr}(c \mid s) = \frac{2}{(\mid s \mid +1)(\mid s \mid +2)} \times \sum_{i=0}^{\mid s \mid} \left(\sum_{k=0}^{\mid s \mid - i} \frac{(\varepsilon / (\mid \mathscr{A} \mid -1))^{d_{i,k}} (1-\varepsilon)^{\mid s \mid - i - k - d_{i,k}}}{\mid \mathscr{A} \mid^{i+k}} \right)$$

where $d_{l,k}$ is the Hamming distance between substring$_{l,k}(c)$ and substring$_{l,k}(s)$.

- Trace Reconstruction Problem in the *Mutate$_\varepsilon$(TrimAndExtend)* model
- **Input:** A trace-set C generated from an unknown seed string according to the *Mutate$_\varepsilon$(TrimAndExtend)* model.
- **Output:** A string maximizing $\mathbf{Pr}(C \mid s)$.

Extend$_t$(Mutate$_\varepsilon$(Trim)): The biologically adequate model for generating traces from a seed string is the *Extend$_t$(Mutate$_\varepsilon$(Trim))* model illustrated in Figure 3. This model is more complex than the previous ones as it requires consideration of all possible pairs of equally sized substrings of the seed string and the trace. Note that in all previous models, the traces either had the same length as the seed string, or were aligned with the seed string on the left. Let sub$^l(s)$ denote all the substrings of s of length l and $\mathrm{sub}_t^l(c)$ denote all substrings of c of length l such that the number of symbols in c before or after the substring do not exceed t. Then, the probability that a seed string s generates a trace c in the *Extend$_t$(Mutate$_\varepsilon$(Trim))* model is given by

$$\mathbf{Pr}(c \mid s) = \frac{1}{(t+1)^2} \frac{2}{(|s|+1)(|s|+2)} \times \sum_{l=0}^{\min(|s|,|c|)} \frac{1}{|\mathscr{A}|^{|c|-l}} \left(\sum_{\substack{\bar{s} \in \mathrm{sub}^l(s) \\ \bar{c} \in \mathrm{sub}_t^l(c)}} (1-\varepsilon)^{l-d_{\bar{s},\bar{c}}} \left(\frac{\varepsilon}{|\mathscr{A}|-1} \right)^{d_{\bar{s},\bar{c}}} \right), \tag{4}$$

where d_{s_1,s_2} is the Hamming distance between strings s_1 and s_2.

- Trace Reconstruction Problem in the *Extend$_t$(Mutate$_\varepsilon$(Trim))* model
- **Input:** A trace-set C generated from an unknown seed string according to the *Extend$_t$(Mutate$_\varepsilon$(Trim))* model.
- **Output:** A string maximizing $\mathbf{Pr}(C \mid s)$.

C. Trace Reconstruction of V, D, and J genes

Above, we considered the trace reconstruction problems that are relevant to generating a CDR3 from a D gene. We will now consider more complex trace reconstruction problems that model concatenation of V, D, and J genes to form an entire immunoglobulin gene (Figure 6). We will start from the simplest problem when each trace represents a concatenation of just two traces generated by two different seed strings.

SuffixExtend$_t$(TrimSuffix)*SuffixExtend$_t$(TrimSuffix): We first consider a model when two seed strings s_1, s_2 of equal length n generate a single trace c according to the *SuffixExtend$_t$(TrimSuffix(s_1))* SuffixExtend$_t$(TrimSuffix(s_2))* model (Figure 6a). Let prefix$_l$(s) and suffix$_l$(s) be the prefix and suffix of string s of length l. The probability that the seed strings s_1 and s_2 generate a trace c is given by

$$\mathbf{Pr}(c \mid s_1, s_2) = \sum_{l=0}^{|c|} \mathbf{Pr}(\mathrm{prefix}_l(c) \mid s_1) \times \mathbf{Pr}(\mathrm{suffix}_{|c|-l}(c) \mid s_2) \tag{5}$$

where $\mathbf{Pr}(\mathrm{prefix}_l(c) \mid s_1)$ is defined according to the *SuffixExtend$_t$(TrimSuffix)* model (Eq. 3) if $l \le n + t$ and 0 otherwise. $\mathbf{Pr}(\mathrm{suffix}_{|c|-l}(c) \mid s_2)$ is defined similarly.

- Trace Reconstruction Problem in the *SuffixExtend$_t$(TrimSuffix)* SuffixExtend$_t$(TrimSuffix)* model
- **Input:** A trace-set C generated from two unknown seed strings according to the *SuffixExtend$_t$(TrimSuffix)* SuffixExtend$_t$(TrimSuffix)* model.
- **Output:** Strings s_1 and s_2 maximizing $\mathbf{Pr}(C \mid s_1, s_2)$.

SuffixExtend$_t$(TrimSuffix)*SuffixExtend$_t$(TrimSuffix) model with multiple seeds: Next, we consider a modification of the above model where each trace is generated by two sets of seed strings of the same length n, $S_1 = \{s_1^1, s_1^2, \ldots, s_1^{M_1}\}$ and $S_2 = \{s_2^1, s_2^2, \ldots, s_2^{M_2}\}$, rather than a pair of seed strings. Seed strings s_1 and s_2 are randomly chosen (from the sets S_1 and S_2 according to a uniform distribution) and the chosen strings generate a trace according to the *SuffixExtend$_t$(TrimSuffix)* SuffixExtend$_t$(TrimSuffix)* model.

- Trace Reconstruction with Multiple Seeds Problem in the *SuffixExtend$_t$(TrimSuffix)* SuffixExtend$_t$(TrimSuffix)* model

- **Input:** A trace-set C generated from two unknown sets containing M_1 and M_2 seed strings according to the *SuffixExtend$_t$(TrimSuffix)* SuffixExtend$_t$(TrimSuffix)* model.

- **Output:** A set of M_1 seed strings and a set of M_2 seed strings maximizing $\mathbf{Pr}(C \mid S_1, S_2)$.

VDJ recombination model (single v, d, and j seed strings): We now consider a model when three strings v, d, and j of length n_v, n_d, and n_j respectively generate a trace c according to the *Mutate$_\varepsilon$(TrimSuffix(v)* Extend$_t$(Trim(d))* TrimPrefix(j))* model (Figure 6b). Here, *TrimPrefix(s)* is defined similarly to *TrimSuffix(s)*, where an integer k is sampled uniformly from $[0, |s|]$, and the prefix of s of length k is trimmed. However, like the *Extend$_t$(Mutate$_\varepsilon$(Trim)))* model, it is a complicated model because one must consider all triples of substrings of the trace c. The probability $\mathbf{Pr}(c \mid v, d, j)$ that the seed strings v, d, and j generate a trace c is given by

$$\mathbf{Pr}(c \mid v, d, j) = \sum_{i=0}^{n_v} \sum_{k=0}^{\min(|c| - i, n_j)} P_1(\text{prefix}_i(c) \mid v) \times P_2(\text{substring}_{i,k}(c) \mid d) \times P_3(\text{suffix}_k(c) \mid j),$$

where P_1 (prefix$_i(c) \mid v$) is given by

$$P_1(\text{prefix}_i(c) \mid v) = \frac{1}{n_v + 1}(\varepsilon / (|\mathscr{A}| - 1))^{d_i}(1 - \varepsilon)^{i - d_i}$$

where d_i is the Hamming distance between prefix$_i(c)$ and prefix$_i(v)$. P_2(substring$_{i,j}(c) \mid d$) is defined as in Eq. (4). P_3 is defined similarly to P_1.

- Trace Reconstruction Problem in the *VDJ recombination* (single v, d, and j seed strings) model

- **Input:** A trace-set C generated from three unknown seed strings according to the *VDJ recombination* model.

- **Output:** Three strings s_1 , s_2, and s_3 maximizing $\mathbf{Pr}(C \mid v, d, j)$.

VDJ recombination model (multiple v, d, and j seed strings): We will now consider a model when three seed-sets $V = \{v_1, v_2, \ldots, v_{M_V}\}$, $D = \{d_1, d_2, \ldots, d_{M_d}\}$, and $J = \{j_1, j_2, \ldots, j_{M_j}\}$ generate a trace c according to the following model. One string from each of the sets V, D, and J is uniformly randomly chosen and the chosen strings v, d, and j generate a trace according to the VDJ recombination model. The probability that a trace c is generated by seed strings in V, D, and J is given by

$$\mathbf{Pr}(c \mid V, D, J) = \frac{1}{M_v M_d M_j} \sum_{v \in V} \sum_{d \in D} \sum_{j \in J} \mathbf{Pr}(c \mid v, d, j)$$

- Trace Reconstruction Problem in the *VDJ recombination* (multiple v, d, and j seed strings) model
- **Input:** A trace-set C generated from three unknown seed-sets (containing M_v, M_d, and M_j strings respectively) according to the *VDJ recombination* model.
- **Output:** Set S_1 with M_v strings, set S_2 with M_d strings, and set S_3 with M_j strings maximizing $\mathbf{Pr}(C \mid S_1, S_2, S_3)$.

IV. Trace Reconstruction problems for DNA Data storage

A popular formulation of trace reconstruction considers the *deletion channel*, where random symbols in the seed string s are deleted independently with probability q and $0 < q < 1$ is the *deletion probability*. This produces a trace c representing a random subsequence of s. This process is repeated independently T times to produce a random trace-set C (Figure 7). The trace reconstruction algorithm takes the traces (without any information about which symbols were deleted from the seed string), the length of the seed string, and the deletion probability as an input. For simplicity, we focus on binary seed strings, while the definitions can be extended to larger alphabets.

The maximum likelihood solution would output the string s that maximizes $\mathbf{Pr}(C \mid s)$ for the given trace-set C. We first consider the probability $\mathbf{Pr}(c \mid s)$ for a single trace c. Let $N_s(c)$ denote the number of times c appears as a subsequence of s. For example, if $s = 11010$ then $c = 110$ appears $N_s(c) = 4$ times, corresponding to the subsequences {110••, 11•• 0, 1•• 10, • 1 • 10}, where • denotes a deleted symbol. The value of $N_s(c)$ can be computed using dynamic programming [21]. Recalling that $|s|$ denotes the length of a string, the probability $\mathbf{Pr}(c \mid s)$ can be computed as follows

$$\mathbf{Pr}(c \mid s) = N_s(c) \cdot q^{|s| - |c|} (1 - q)^{|c|} .$$

Since each trace in C is produced independently, we have that

$$\mathbf{Pr}(C \mid s) = \prod_{c \in C} \mathbf{Pr}(c \mid s) .$$

The value $\mathbf{Pr}(C \mid s)$ can be calculated for any fixed s. However, the optimization problem that determines the optimal s is challenging. Designing an efficient algorithm (with time polynomial in $|C|$ and $|s|$) that outputs a string s maximizing $\mathbf{Pr}(C \mid s)$ is an open question. Partial results are known when $|C|$ is very small [19], [91]–[94].

We focus on the success probability in this section, and we also restrict to length n seed strings. We define the worst-case success probability of an algorithm A over all binary strings of length n as

$$P_A(n, T) = \min_s P_A(s, T)\,.$$

Similarly, the average-case success probability of A over all binary strings of length n is

$$\widetilde{P}_A(n, T) = \frac{1}{2^n} \cdot \sum_s P_A(s, T)\,.$$

Trace Complexity:

Most previous work provides information-theoretic results in terms of the *trace complexity*, which is the minimum value of T such that there exists an algorithm with success probability at least the given *ReconstructionRate*. This will depend on the deletion probability q. For any fixed *ReconstructionRate*, the number of input traces must be at least the trace complexity for the algorithmic problem to be feasible. It is often convenient to fix the *ReconstructionRate* to a default value, such as *ReconstructionRate* = 0.95. This does not affect the trace complexity too much because arbitrarily large *ReconstructionRate* can be achieved by increasing the number of traces by a logarithmic factor (taking a majority vote over several trials). Therefore, we define the *worst-case trace complexity* as

$$T_q(n) = \arg\min\left\{T \mid \max_A P_A(n, T) \geq 0.95\right\}$$

and the *average-case trace complexity* as

$$\widetilde{T}_q(n) = \arg\min\left\{T \mid \max_A \widetilde{P}_A(n, T) \geq 0.95\right\}.$$

The trace complexity may depend on the error rate. Certain algorithms only succeed when the deletion probability decreases as a function of the length n of the seed string. Historically, the initial results assume that the deletion probability scales inversely with n, e.g., $q = O(1/\sqrt{n})$ or $q = O(1/\log n)$ [4], [12], [20]. These results have been later strengthened to handle a constant rate of deletions, e.g., $q = 0.5$ [7], [13], [18]. The extent to which the deletion probability impacts the trace complexity remains unknown in general.

For simplicity, we restrict our attention to the deletion channel, but many of the results that we discuss also extend to a more general error model that includes insertions and substitutions [7], [13], [18], [20]. We refer the reader to the following surveys for other error models and related theoretical open questions [91], [95].

V. Theoretical Results on Trace Reconstruction

We survey theoretical results for reconstructing a seed string s of length n. We begin with three variants depending on the nature of the unknown string: it can be arbitrary (worstcase); it can be chosen uniformly at random (average-case); or, it can be chosen from a predefined set of encoded strings (*coded trace reconstruction*). For each variant, we first present a formal problem statement. The information-theoretic goal is to determine the values of the parameters T, q, n, and *ReconstructionRate* for which the problem is solvable. The next step is to design an efficient algorithm for such cases. In the latter half of this section, we also mention generalizations to multiple strings and to higher-order structures (such as trees). We conclude with a brief description of some recent practical developments. Throughout, we use $\hat{s} = A(C)$ to abbreviate the output of a reconstruction algorithm A on an input trace-set C.

Worst-case trace reconstruction:

We first describe the case where the seed string s is arbitrary, and the success probability is calculated over the randomness in generating the trace-set C.

Worst-Case Trace Reconstruction Problem for the Deletion Channel

Input: A random trace-set C of size T generated from a seed string s of length n according to the deletion channel model with deletion probability q, as well as the *ReconstructionRate*.

Output: A string $\hat{s}$ such that $\hat{s} = s$ with success probability at least *ReconstructionRate*.

The current best trace complexity for worst-case strings is $T_q(n) = \exp(O(n^{1/5} \log^5 n))$ when the deletion probability q is at most 1/2 [96]. When $q \in (1/2, 1)$, then the known result is $T_q(n) = \exp(O(n^{1/3}))$ [7], [18]. The latter result uses a *mean-based algorithm* that first pads each trace with trailing zeros so that the length equals the seed length n (here, we consider a binary alphabet). Then, the mean of the traces is computed by summing the padded traces coordinate-wise and normalizing by the number of traces (i.e., this computes the fraction of ones in each position). It is known that when the number of traces is at least $\exp(O(n^{1/3}))$ then these means suffice to determine the unknown string with high success probability [7], [18]. The improvement to $T_q(n) = \exp(O(n^{1/5} \log^5 n))$ when $q \leq 1/2$ uses a similar algorithm, with the subtle difference and important difference that certain substring frequencies are approximated instead of single bits [96].

An intriguing aspect of the worst-case result is the use of techniques from complex analysis. The elegant argument involves expressing the mean-based statistics (from averaging the padded traces) in terms of a complex-valued generating function (whose coefficients are determined by the seed string and deletion probability). The aim is to lower bound the statistical distance between trace-sets that are generated from distinct seed strings. It is fairly easy to show that the maximum modulus of this function in a certain arc of the complex unit disk provides such a lower bound. Then, to complete the proof, the authors use prior results on Littlewood polynomials [97], [98]. This argument serves as the basis of a trace reconstruction algorithm with running time proportional to the number of traces. Surprisingly, the bound is tight for mean-based algorithms, in the sense that $\exp(\Omega(n^{1/3}))$

traces are necessary if an algorithm uses only the coordinate-wise means [7], [18]. These results have further inspired the use of related generating functions to derive improved bounds for other statistical learning problems [99], [100].

Improvements to the trace complexity are known for a very small deletion probability; if each bit is deleted with probability less than $n^{-1/2-\delta}$ for a small constant δ, then a nearly-linear number of traces suffice [12]. We note that mean-based algorithms extend to handle insertions and substitutions as well [7], [18]. It is an open question to determine the smallest deletion probability such that a polynomial number of traces suffice. When the deletion probability does not decrease with n (e.g., $q = 0.5$), then lower bounds on the trace complexity are known. Previous work shows $T_{0.5}(n) = \widetilde{\Omega}(n^{3/2})$ traces are necessary [8], [11], where the $\widetilde{\Omega}$ notation hides polylog factors.

The central open problem is to close the exponential gap between upper and lower bounds on the worst-case trace complexity. A first step could be to better understand which seeds strings are the most challenging to reconstruct. For many algorithms, simple strings demonstrate that the current analysis is tight. However, other methods readily reconstruct these strings. For example, the $\widetilde{\Omega}(n^{3/2})$ lower bound is derived for the task of distinguishing a pair of alternating strings with two flipped bits, e.g..

- $1010 \cdots 10\underline{10}10 \cdots 1010$
- $1010 \cdots 10\underline{01}10 \cdots 1010$

Telling apart these strings using traces is straightforward, and an algorithm using $\widetilde{O}(n^{3/2})$ traces is known. Hence, the lower bound for this pair is nearly tight [8], [11]. Another futile attempt comes from considering a uniformly random string. In many areas, the probabilistic method suffices to identify difficult instances [101], [102]. For reconstruction problems, the opposite is often true: random objects can be reconstructed with less information than worst-case instances [103]-[105]. In particular, random strings are easier to reconstruct, as we will now see.

Average-case trace reconstruction:

We move on to consider the case when the seed string s is a uniformly random length n string. In this case, the seed string is chosen randomly before generating each set of traces, and the success probability is calculated with respect to both the trace-set generation and the choosing of the seed string.

- Average-Case Trace Reconstruction Problem for the Deletion Channel
- **Input:** A random trace-set C of size T generated from a uniformly random seed string s of length n according the deletion channel model with deletion probability q, as well as the *ReconstructionRate*.
- **Output:** A string $\hat{s}$ such that $\hat{s} = s$ with success probability at least *ReconstructionRate*.

The current best upper bound on the trace complexity is $\widetilde{T}_q(n) = \exp(O(\log^{1/3} n))$ for uniformly random strings, and this holds for any deletion probability q bounded away from one [13]. This upper bound is exponentially better than the result for worst-case strings [7], [18]. The lower bound for average-case reconstruction shows that $\widetilde{T}_{0.5}(n) = \widetilde{\Omega}(\log^{3/2} n)$ traces are necessary to reconstruct a random string with constant deletion probability, where here the $\widetilde{\Omega}$ notation hides log log n factors [8], [11]. When the deletion probability scales inverse-logarithmically with n, then logarithmic upper bounds on the average-case trace complexity are known [4], [20].

The algorithms for average-case reconstruction are much more involved than the current methods for worst-case reconstruction. Instead of relying only on statistical quantities, the algorithm iteratively reconstructs the seed string one character at a time. At the beginning, a small number of traces are used to learn a short prefix exactly. This partial reconstruction then serves as an anchoring method to approximately align the traces. When the seed string is random, its short substrings are locally unique with high probability, and therefore, such alignments can be reliable. The algorithm moves left-to-right and employs a worst-case algorithm to reconstruct the next bit. This general approach, along with a careful analysis of the alignment process, led to an algorithm that requires $\exp(O(\sqrt{\log n}))$ traces when the deletion probability is less than 0.5 [106], building on a similar approach that uses poly(n) traces [12]. Subsequent work extends this idea with a more sophisticated alignment method and many technical developments, leading to the best known algorithm for average-case trace reconstruction that achieves a trace complexity of $\widetilde{T}_q(n) = \exp(O(\log^{1/3} n))$ for any deletion probability q bounded away from one [13]. Recently, an algorithm has also been proposed that achieves a polynomial number of traces in a *smoothed-analysis* setting that interpolates between the worst-case and average-case reconstruction problems; more specifically, in this model, a worst-case seed string is first randomly perturbed, where each bit is flipped with some probability *less than* 1/2, and then the traces are all generated from this randomized string [107].

Coded trace reconstruction:

The next variation assumes that the seed string s is chosen from a predefined set of possible strings (e.g., these may be *codewords* from a suitable code, where it is desirable for these codewords to have an efficient construction procedure as well). For example, in DNA data storage, there is flexibility to encode the seed strings. The definition of success probability can either be the minimum over all predefined seed strings (worst-case) or the expectation over a uniformly random predefined seed string (average-case).

- Coded Trace Reconstruction Problem for the Deletion Channel
- **Input:** A random trace-set C of size T generated from a seed string s of length n according to the deletion channel model with deletion probability q, where s is guaranteed to be from a predefined set of possible strings, as well as the *ReconstructionRate*.
- **Output:** A string $\hat{s}$ such that $\hat{s} = s$ with success probability at least *ReconstructionRate*.

Compared to reconstructing worst-case strings, better trace complexity upper bounds are known. The improvement depends on the number of possible encoded strings, i.e., the rate of the code [5], [6], [9]. We mention a few results that exemplify different regimes. For this discussion, we consider worst-case reconstruction, where the success probability guarantee holds for all predefined strings. It will also be convenient to frame the encoding process as adding redundancy to an arbitrary seed string. The code maps the unknown seed string s of length n to a new string s' of larger length $n' > n$. Applying this mapping to all possible strings generates the predefined seed strings in the coded trace reconstruction problem. The objective is to simultaneously minimize n' while developing an efficient reconstruction algorithm with small trace complexity.

We say the code has *redundancy* $n' - n$ equal to the number of extra characters in the encoding. When the redundancy is small, such as $O(n/\log n)$, algorithms are known with trace complexity polylog(n), which is sublinear in seed string length [9]. The high-level strategy is to create the new string s' by concatenating many codewords. The added redundancy comes from padding the codewords with a run of zeros followed by a run of ones. For example, the codewords could have length $\Theta(\log^2 n)$ and runs have length $\Theta(\log n)$. This implies that none of the padded portions are deleted in a trace with high probability. The padding enables the algorithm to align the codeword portions in each trace. The redundancy for such a scheme is $O(n/\log n)$. After identifying the padded and codeword portions, the encoded seed string s' can be reconstructed from polylog(n) traces.

In the larger redundancy regime, such as redundancy εn with $\varepsilon \in (0, 1)$ being a constant, an improved trace complexity of $\exp(O(\log^{1/3} (1/\varepsilon)))$ is achievable [6]. Recent work also more thoroughly studies coded trace reconstruction in the insertion/deletion channel when there are a constant number of errors or a constant number of traces [5], [92], [108]-[110]. Before integrating these results into a DNA data storage system, certain ulterior constraints should be addressed as well. The synthesis process imposes limitations on the seed string length, and hence, the redundancy must be relatively small [24], [30]. Trace reconstruction is also only one part of the pipeline. The encoding and decoding schemes may need to satisfy other properties, such as error-correction capabilities [30] and enough separation between seed strings to enable clustering [46].

Non-uniform error rate:

The deletion channel model assumes that the deletion probability q is fixed for all characters in the seed string. A biologically relevant modification considers varying deletion probabilities, where the position or value of each character may affect the error probability [10]. For certain assumptions on the deletion probabilities, the current best algorithm is the same as for worst-case strings with constant deletion probability (i.e., a mean-based algorithm), and the trace complexity is asymptotically the same $\exp(O(n^{1/3}))$ as well. It is an important open question to extend current theoretical results to more realistic error models.

A. Reconstruction of multiple seed strings

In many applications, the goal is to reconstruct a set of unknown seed strings (rather than a single seed string) given a set of their traces. For example, in DNA data storage, the original set of short seed strings is stored together as an unordered collection in a tube. Recovering the data results in a set of traces arising from these seed strings and involves accurately determining a large fraction of the seed strings. Storing and retrieving a set of strings leads to interesting coding-theoretic problems as well [56], [59], [61], [63], [64], [68], [69].

Trace reconstruction for multiple strings has been explored recently [111]-[113]. Historically, this originates in the area of *population recovery*, determining an unknown distribution over a set of strings [114], [115]. In the language of trace reconstruction, the population recovery model can be described as follows. There is an unknown set S of seed strings, where only the number of strings in S is given as an input. The traces are generated using a two-step process. First, a string s is chosen randomly from the set of seed strings S based on the uniform distribution over S. Then, a trace is produced from s. This process repeats T times, leading to a trace-set C. The goal is to reconstruct at least a $1 - \delta$ fraction of the strings in S for a given accuracy parameter $0 < \delta < 1$. In other words, the algorithm outputs a candidate set $\hat{S}$ with $|\hat{S}| = |S|$, and we require that $|\hat{S} \cap S| \geq (1 - \delta)|S|$. The success probability is defined as the probability that $|\hat{S} \cap S| \geq (1 - \delta)|S|$, calculated over the random trace-set.

Analogous to the single string problems, there are variations depending on whether a set of seed strings is an arbitrary (worst-case) or random (average-case) set of strings [111]-[113]. For the worst-case version, we define the success probability over the randomness in the trace-set generation. For the average-case version, we also include the probability of choosing random set S of length n strings where $|S|$ is fixed. We remark that prior work actually considers a more intricate population recovery model for a non-uniform distribution over S [111], [112], [114], [115]. However, we use the uniform distribution because it seems more relevant to practical applications (e.g., in DNA data storage, the seed strings are chosen from S with approximately equal probability).

- Multiple String Trace Reconstruction Problem for the Deletion Channel
- **Input:** A random trace-set C of size T generated from a set of unknown seed strings S of length n according the Deletion Channel model with deletion probability q and an accuracy parameter δ, as well as the *ReconstructionRate*.
- **Output:** A set of strings $\hat{S}$ with $|\hat{S}| = |S|$ such that $|\hat{S} \cap S| \geq (1 - \delta)|S|$ with success probability at least *ReconstructionRate*.

The output is verifiable when the original set of strings S is known. In DNA data storage, the set S corresponds to the set of strings that store the data, which may be used to benchmark a reconstruction algorithm.

Average-case population recovery problem has a straightforward reduction to the single string case, both in theory [112] and in practice [30], [46]. When the seed strings are sufficiently long, they are also far apart geometrically because they have pairwise edit distance scaling linearly with their length [47]-[49]. This ensures a clear separation between groups of traces that come from one seed string rather than another. Clustering methods can accurately partition the trace-set into subsets that are generated from each individual seed string [46], [112]. Then, algorithms for the average-case problem will succeed in exactly reconstructing most of the seed strings from the clusters. When there are $|S| = M$ seed strings, the trace complexity is poly(M) · exp($O(\log^{1/3} n)$) [112].

Reconstructing a worst-case collection of seed strings is more challenging. The first approach to do so rigorously relied on subsequence statistics, and their method uses $\exp(n^{O(M)} \cdot \sqrt{n})$ traces [111]. Subsequent work improved this bound by showing how to extend the mean-based analysis for the worst-case reconstruction of a single seed string [113]. The resulting algorithm uses only exp($O(M^3 \cdot n^{1/3})$) traces. Notice that when $M = 1$, then this matches the best known bound for a single worst-case string [7], [18].

B. Reconstructing Higher-Order Structures

Recent work proposes a generalization of string trace reconstruction, known as *tree trace reconstruction* [116]. The goal is to reconstruct a node-labeled tree using traces from a channel that deletes nodes. The tree topology is known ahead of time, and learning the unknown node labels is the sole objective. They propose two deletion models that differ from each other based on how the children of a deleted node move in the tree. Figure 8 depicts an example tree and trace for one of the models, which is derived from the notion of tree edit distance. When a node is deleted, its children move up to become children of the deleted node's parent. In particular, deletions still result in a connected tree. For technical reasons, the root is never deleted. The model assumes a left-to-right ordering of every level, and hence, the trees are presented in a consistent way. The tree reconstruction problem in this model generalizes string reconstruction from the deletion channel, coinciding when the tree is a path.

The tree reconstruction problem provides a vantage point to study the complexity of reconstructing higher-order structures. Perhaps surprisingly, for many classes of trees, such as complete k-ary trees and multi-arm stars (a.k.a. *spider trees*), a polynomial number of traces suffice for worst-case reconstruction [116]. This is in contrast to the string case, where the current algorithms use exponentially many traces [7], [18]. The algorithms for reconstructing complete k-ary trees also differ significantly from the known methods for string reconstruction. As there is more structure in the tree, combinatorial methods can be used to identify the location of certain subtrees. The algorithms make heavy use of traces that contain a root-to-leaf path of the same length as the depth of the seed tree. If the deletion probability is constant, and the tree has depth $O(\log n)$, then such a path survives with inverse-polynomial probability. Under certain conditions, the nodes in such paths suffice to recover the corresponding labels. The algorithm for reconstructing spider trees proceeds via a mean-based approach (analogous to the worst-case reconstruction results [7], [10], [18]). This involves generalizing the complex-analytic techniques to capture mean-based statistics for spider trees. It also is known that paths (a.k.a. strings) are the most difficult tree because any tree can be reconstructed using a string reconstruction algorithm with the same asymptotic trace complexity. Related endeavors study reconstructing matrices from a channel that deletes rows and columns [14] or circular seed strings from a channel that applies a random circular permutation before deleting characters [117].

Biological motivation for the tree trace reconstruction problem can be loosely attributed to the goal of identifying certain molecules that inherently have a tree-like structure. For example, recent advances have shown that tree-structured DNA is useful for bio-sensing applications [118], [119] and storing digital information [120]. In these applications, a variety of tree topologies have been studied. The DNA molecule could take a star-shaped form, with multiple *arms* connected to a shared center. The arms may be single- or double-stranded DNA, and each arm of the star contains roughly 50–100 nucleotides. Such nanostructures have been developed in the context of DNA-based nanomaterials [121], using building blocks such as a 4-arm star, known as a *Holliday junction* [122].

The tree trace reconstruction problem arises when sequencing such tree-structured DNA. More specifically, a potential objective could be to efficiently verify that a constructed molecule has the intended shape. Nanopore devices may be able to sequence tree-structured DNA directly, providing reads that resemble traces in the tree reconstruction model. Promising initial results have been obtained for sequencing Y-shaped and T-shaped DNA [119], as well as extensions to stars with up to twelve arms and certain DNA hairpin structures [118], [120].

C. Practical Trace Reconstruction Solutions

Many theoretical trace reconstruction algorithms assume that the number of traces and the length of the seed string are both unrealistically large. Coded trace reconstruction is an exception, where simple algorithms are known with sublinear trace complexity and running time. It is possible that these theoretical algorithms can be used in practical DNA data storage systems to improve the trace complexity. A remaining challenge is combining the codes for trace reconstruction with the codes for error-correction, which is an interesting avenue for future work.

Adapting the current best theoretical algorithms for worst-case or average-case reconstruction into practical solutions seems unlikely. Instead, a promising direction is to use alignment-based methods, such as bitwise-majority alignment [4]. These perform well for the average-case problem when the deletion probability is small, and they can be efficiently implemented in near linear time. In one DNA data storage system, this has been successfully used when combined with certain undisclosed heuristics [30]. The idea is to start with a pointer at the beginning of each trace and move left-to-right. At each position, a majority vote is taken to determine the most likely symbol in that position. This majority symbol will be the output value of that position. Then, the pointers must be updated. If a trace agrees with the majority, then its pointer is advanced to the right by one. For the disagreeing traces, other methods must be used to guess whether the error was due to an insertion, substitution, or deletion. It is often beneficial to look ahead to the next few positions to help guess the type of error (e.g., if the next bit agrees with the majority, then the error is more likely to have been a substitution than a deletion). Depending on the type of error, the pointers for the disagreeing traces are moved appropriately.

A related approach uses a multi-sequence alignment method [123] in conjunction with majority voting and certain preprocessing steps [33]. Especially error-ridden traces are discarded before reconstruction. This can be based on simple criteria, such as the length of the trace or the correctness of the address portion (e.g., if the DNA primer is intact). More sophisticated methods may be used depending on the error-correcting code (e.g., parity or cyclic redundancy checks). Discarding many traces incurs a higher cost of sequencing and reconstruction, and therefore, it would be better to selectively use certain traces at different steps of the reconstruction process. The desired *ReconstructionRate* depends on the redundancy in the error-correcting codes [30], [33].

Recent works have taken a different approach and developed ways to approximate the maximum likelihood solution [19], [92]-[94]. The focus here has been on developing algorithms that approximately reconstruct the seed string when given a small budget on the number of traces (e.g., 2–10). In some cases, these techniques outperform statistical and alignment-based approaches. While this progress is promising, it is still largely an open problem to design efficient algorithms that achieve a high *ReconstructionRate* with a small number of traces.

VI. Conclusion

In this review paper, we discussed applications of the Trace Reconstruction Problem in immunogenomics and DNA data storage. We introduced new trace generation models, presented a variety of open questions, surveyed existing solutions, and discussed their applicability and shortcomings. Given that computational immunogenomics and DNA data storage are young and rapidly expanding research areas, we expect more theoretical techniques, algorithms, and publicly available datasets to emerge in the next several years.

We close with a summary of some key open questions along with general perspectives.

- **Maximum Likelihood vs. Trace Complexity:** Sections III and V address different objectives. What are the key similarities and differences between the maximum likelihood solution and the maximum success probability solution? When does a budget on the number of traces radically influence the best reconstruction algorithm? Is there a gap between the trace complexity for computationally efficient vs. information-theoretic reconstruction?
- **Immunogenomics Models:** Throughout Section III we have introduced several trace generation models that vary in terms of their complexity and realism. Given that these models have yet to be seriously studied, many open questions remain. Can we design polynomial-time algorithms for computing the maximum likelihood solution? Can we derive tight bounds on the trace complexity for information-theoretic reconstruction?
- **Practical Implementations:** It remains to be seen whether an improved theoretical understanding of trace reconstruction algorithms will lead to effective empirical solutions. In Section V-C, we have briefly addressed some of the known practical algorithms for the deletion channel. What are the best performing methods in practice, in terms of trace complexity, success probability, running time, and generality? If we want to experimentally test various algorithms, what are the important properties of benchmark datasets?
- **Confidence Measures:** Another desirable property for both immunogenomics and DNA data storage applications would be to output a measure of confidence

in the reconstructed string. Is it the case that most seed strings are easy to reconstruct in practice, while only a small set of strings and traces are challenging?

- **Data Driven Models:** The models that we have surveyed involve various parameters that determine the error rate in the trace generation process. Can we experimentally determine these parameter values? Is it possible to optimize the reconstruction algorithm for the most prevalent error rates and the most realistic models?
- **Approximate Reconstruction:** The formulation of success probability in Sections II and IV hinge on the requirement that the seed string is exactly reconstructed. Can we design algorithms that use fewer traces and output a candidate string within a small edit or Hamming distance of the seed string? If these additional errors can be handled with error-correcting codes, then how do approximate reconstruction algorithms compare to other approaches for coded trace reconstruction?
- **End-to-end Solutions.** Production-level DNA data storage systems will involve a co-design of the core pipeline components. Can we develop an encoding scheme that enables efficient trace reconstruction and clustering, while also providing error-correcting capabilities and high storage density?

Acknowledgments

We thank the anonymous reviewers for helpful comments on earlier versions of this manuscript. We thank Sami Davis, Rex Lei, Miklós Z. Rácz, João Ribeiro, Omer Sabery, and Eitan Yaakobi for helpful discussions.

Funding

The work of VB was supported by the Qualcomm Institute at UCSD. The work of PAP was supported by the NIH 2-P41-GM103484PP grant and the NSF EAGER award 2032783. The work of CR was supported by the UCSD Data Science Fellowship 2019–2020. The work of YS was supported by the AAI Intersect Fellowship 2019 and the NSF EAGER award 2032783.

Biography

Vinnu Bhardwaj is a PhD candidate in the department of Electrical and Computer Engineering at University of California, San Diego (UCSD), with specialization in data science and machine learning. Prior to UCSD, he received his ME in ECE from the Indian Institute of Science (IISc), and his BE from PEC University of Technology, India.

His research interests include the development of computational methods to better understand biological mechanisms using data in different domains including immunogenomics and metabolomics. He is the author of MINING-D, the tool that lead to the discovery of 25 novel IGHD genes. He was awarded with the Dean’s Office Fellowship by UCSD (2015).

Pavel A. Pevzner is Ronald R. Taylor Professor of the Computer Science and Engineering and Director of the NIH Center for Computational Mass Spectrometry at University of

California, San Diego. He holds Ph.D. from Moscow Institute of Physics and Technology, Russia. He was named Howard Hughes Medical Institute Professor in 2006.

He was elected the Association for Computing Machinery Fellow in 2010, the International Society for Computational Biology Fellow in 2012, the European Academy of Sciences member (Academia Europaea) in 2016, and the American Association for Advancement in Science (AAAI) Fellow in 2018. He was awarded a Honoris Causa (2011) from Simon Fraser University in Vancouver, the Senior Scientist Award (2017) by the International Society for Computational Biology, and the Kanellakis Theory and Practice Award from the Association for Computing Machinery (2019).

Dr. Pevzner authored textbooks “Computational Molecular Biology: An Algorithmic Approach”, “Introduction to Bioinformatics Algorithms” (with Neal Jones), Bioinformatics Algorithms: an Active Learning Approach (with Phillip Compeau), and Learning Algorithms through Programming and Puzzle Solving (with Alexander Kulikov). He co-developed the Bioinformatics and Data Structure and Algorithms online specializations on Coursera as well as the Algorithms Micro Master Program at edX.

Cyrus Rashtchian is currently a Data Science Fellow at the University of California, San Diego, affiliated with the Computer Science & Engineering department and the Qualcomm Institute. He received his Ph.D. in Computer Science & Engineering in 2018 from the University of Washington, Seattle, and his BS in Computer Science in 2010 from the University of Illinois, Urbana-Champaign.

His broad research interests are motivated by building the foundations of data science, including DNA data storage, robust and explainable machine learning, computational and statistical trade-offs, distributed algorithms, and clustering. In general, he applies diverse geometric and algorithmic tools to problems in data science, with a keen eye for new applications and emerging technologies. Prior to UCSD, he has completed research internships at Facebook Reality Labs, Microsoft Research, and Cray. He has published in top machine learning and theoretical computer science conferences, including ITCS, SODA, COLT, ICML, NeurIPS, and AISTATS.

Yana Safonova received the B.S. and M.S. degrees in computer science from the Nizhny Novgorod State University, Russia in 2012 and the Ph.D. degree in bioinformatics from the Saint Petersburg State University, Russia in 2017.

Since 2017, she has been a Postdoctoral Researcher at the Computer Science and Engineering Department at University of California, San Diego (UCSD). Since 2019, she has also been affiliated with the Department of Biochemistry and Molecular Genetics at the University of Louisville School of Medicine. Her research interests cover open problems in computational immunology that include applications of the recently emerged immunosequencing technologies to design of antibody drugs, prediction of vaccine efficacy, and population analysis of the immune loci.

Dr. Safonova was awarded with Data Science Postdoctoral Fellowship (2017) by UCSD and Intersect Fellowship for Computational Scientists and Immunologists (2019) by the

American Associations of Immunologists. She is a member of the The Adaptive Immune Receptor Repertoire (AIRR) Community of The Antibody Society.

References

[1]. Levenshtein V, "Reconstruction of objects from a minimum number of distorted patterns," in Doklady Mathematics, vol. 55, no. 3. Pleiades Publishing, Ltd., 1997, pp. 417–420.

[2]. Levenshtein VI, "Efficient reconstruction of sequences," IEEE Transactions on Information Theory, vol. 47, no. 1, pp. 2–22, 2001.

[3]. Levenshtein Vladimir I, "Efficient reconstruction of sequences from their subsequences or supersequences," Journal of Combinatorial Theory, Series A, vol. 93, no. 2, pp. 310–332, 2001.

[4]. Batu T, Kannan S, Khanna S, and McGregor A, "Reconstructing strings from random traces," in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2004, pp. 910–918.

[5]. Abroshan M, Venkataramanan R, Dolecek L, and i Fàbregas AG, "Coding for deletion channels with multiple traces," in 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019, pp. 1372–1376.

[6]. Brakensiek J, Li R, and Spang B, "Coded trace reconstruction in a constant number of traces," arXiv preprint arXiv:1908.03996, 2019.

[7]. De A, O'Donnell R, and Servedio RA, "Optimal mean-based algorithms for trace reconstruction," The Annals of Applied Probability, vol. 29, no. 2, pp. 851–874, 2019.

[8]. Chase Z, "New lower bounds for trace reconstruction," arXiv preprint arXiv:1905.03031, 2019.

[9]. Cheraghchi M, Gabrys R, Milenkovic O, and Ribeiro J, "Coded trace reconstruction," IEEE Transactions on Information Theory, 2020.

[10]. Hartung L, Holden N, and Peres Y, "Trace reconstruction with varying deletion probabilities," in Proceedings of the Fifteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO), 2018, pp. 54–61.

[11]. Holden N, Lyons R et al., "Lower bounds for trace reconstruction," Annals of Applied Probability, vol. 30, no. 2, pp. 503–525, 2020.

[12]. Holenstein T, Mitzenmacher M, Panigrahy R, and Wieder U, "Trace reconstruction with constant deletion probability and related results," in Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2008, pp. 389–398.

[13]. Holden N, Pemantle R, and Peres Y, "Subpolynomial trace reconstruction for random strings and arbitrary deletion probability," in Conference On Learning Theory (COLT), 2018, pp. 1799–1840.

[14]. Krishnamurthy A, Mazumdar A, McGregor A, and Pal S, "Trace Reconstruction: Generalized and Parameterized," in 27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany, ser. LIPIcs, Bender MA, Svensson O, and Herman G, Eds., vol. 144. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 68:1–68:25.

[15]. Magner A, Duda J, Szpankowski W, and Grama A, "Fundamental bounds for sequence reconstruction from Nanopore sequencers," IEEE Transactions on Molecular, Biological and Multi-Scale Communications, vol. 2, no. 1, pp. 92–106, 2016.

[16]. Mao W, Diggavi SN, and Kannan S, "Models and information-theoretic bounds for Nanopore sequencing," IEEE Transactions on Information Theory, vol. 64, no. 4, pp. 3216–3236, 2018.

[17]. McGregor A, Price E, and Vorotnikova S, "Trace Reconstruction Revisited," in European Symposium on Algorithms (ESA). Springer, 2014, pp. 689–700.

[18]. Nazarov F and Peres Y, "Trace Reconstruction with exp(O($N^{1/3}$)) Samples," in Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC), 2017, pp. 1042–1046.

[19]. Srinivasavaradhan SR, Du M, Diggavi S, and Fragouli C, "On maximum likelihood reconstruction over multiple deletion channels," in 2018 IEEE International Symposium on Information Theory (ISIT). IEEE, 2018, pp. 436–440.

[20]. Viswanathan K and Swaminathan R, “Improved String Reconstruction Over Insertion-Deletion Channels,” in Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2008, pp. 399–408.

[21]. Compeau P and Pevzner P, Bioinformatics Algorithms: An Active Learning Approach. Active Learning Publishers, 2018.

[22]. Safonova Y and Pevzner PA, “De novo inference of diversity genes and analysis of non-canonical V (DD) J recombination in immunoglobulins,” Frontiers in immunology, vol. 10, p. 987, 2019. [PubMed: 31134072]

[23]. Bhardwaj V, Franceschetti M, Rao R, Pevzner PA, and Safonova Y, “Automated analysis of immunosequencing datasets reveals novel immunoglobulin D genes across diverse species,” PLoS Computational Biology, vol. 16, no. 4, p. e1007837, 2020. [PubMed: 32339161]

[24]. Ceze L, Nivala J, and Strauss K, “Molecular digital data storage using DNA,” Nature Reviews Genetics, vol. 20, no. 8, pp. 456–466, 2019.

[25]. Church GM, Gao Y, and Kosuri S, “Next-generation digital information storage in dna,” Science, vol. 337, no. 6102, pp. 1628–1628, 2012. [PubMed: 22903519]

[26]. Bornholt J, Lopez R, Carmean DM, Ceze L, Seelig G, and Strauss K, “A DNA-based archival storage system,” in Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, 2016, pp. 637–649.

[27]. Erlich Y and Zielinski D, “DNA Fountain enables a robust and efficient storage architecture,” Science, vol. 355, no. 6328, pp. 950–954, 2017. [PubMed: 28254941]

[28]. Goldman N, Bertone P, Chen S, Dessimoz C, LeProust EM, Sipos B, and Birney E, “Towards practical, high-capacity, low-maintenance information storage in synthesized dna,” Nature, vol. 494, no. 7435, pp. 77–80, 2013. [PubMed: 23354052]

[29]. Meiser LC, Antkowiak PL, Koch J, Chen WD, Kohll AX, Stark WJ, Heckel R, and Grass RN, “Reading and writing digital data in dna,” Nature Protocols, vol. 15, no. 1, pp. 86–101, 2020. [PubMed: 31784718]

[30]. Organick L, Ang SD, Chen Y-J, Lopez R, Yekhanin S, Makarychev K, Racz MZ, Kamath G, Gopalan P, Nguyen B et al., “Random access in large-scale DNA data storage,” Nature Biotechnology, vol. 36, no. 3, p. 242, 2018.

[31]. Shipman SL, Nivala J, Macklis JD, and Church GM, “Crispr–cas encoding of a digital movie into the genomes of a population of living bacteria,” Nature, vol. 547, no. 7663, pp. 345–349, 2017. [PubMed: 28700573]

[32]. Yazdi SHT, Kiah HM, Garcia-Ruiz E, Ma J, Zhao H, and Milenkovic O, “DNA-based storage: Trends and methods,” IEEE Transactions on Molecular, Biological and Multi-Scale Communications, vol. 1, no. 3, pp. 230–248, 2015.

[33]. Yazdi SHT, Gabrys R, and Milenkovic O, “Portable and error-free dna-based data storage,” Scientific reports, vol. 7, no. 1, p. 5011, 2017. [PubMed: 28694453]

[34]. Delves PJ, Martin SJ, Burton DR, and Roitt IM, Essential immunology. John Wiley & Sons, 2017.

[35]. Watson CT and Breden F, “The immunoglobulin heavy chain locus: genetic variation, missing data, and implications for human disease,” Genes & Immunity, vol. 13, no. 5, pp. 363–373, 2012. [PubMed: 22551722]

[36]. Parameswaran P, Liu Y, Roskin KM, Jackson KK, Dixit VP, Lee JY, Artiles KL, Zompi S, Vargas MJ, Simen BB et al., “Convergent antibody signatures in human dengue,” Cell host & microbe, vol. 13, no. 6, pp. 691–700, 2013. [PubMed: 23768493]

[37]. Avnir Y, Watson CT, Glanville J, Peterson EC, Tallarico AS, Bennett AS, Qin K, Fu Y, Huang C-Y, Beigel JH et al., “IGHV1-69 polymorphism modulates anti-influenza antibody repertoires, correlates with IGHV utilization shifts and varies by ethnicity,” Scientific reports, vol. 6, no. 1, pp. 1–13, 2016. [PubMed: 28442746]

[38]. Collins AM, Wang Y, Roskin KM, Marquis CP, and Jackson KJ, “The mouse antibody heavy chain repertoire is germline-focused and highly variable between inbred strains,” Philosophical Transactions of the Royal Society B: Biological Sciences, vol. 370, no. 1676, p. 20140236, 2015.

[39]. Luo S, Jane AY, Li H, and Song YS, “Worldwide genetic variation of the IGHV and TRBV immune receptor gene families in humans,” Life science alliance, vol. 2, no. 2, 2019.

[40]. Yu Y, Ceredig R, and Seoighe C, “A database of human immune receptor alleles recovered from population sequencing data,” The Journal of Immunology, vol. 198, no. 5, pp. 2202–2210, 2017. [PubMed: 28115530]

[41]. Watson CT, Matsen FA, Jackson KJ, Bashir A, Smith ML, Glanville J, Breden F, Kleinstein SH, Collins AM, and Busse CE, “Comment on a database of human immune receptor alleles recovered from population sequencing data,” The Journal of Immunology, vol. 198, no. 9, pp. 3371–3373, 2017. [PubMed: 28416712]

[42]. Potapov V and Ong JL, “Examining sources of error in PCR by single-molecule sequencing,” PloS One, vol. 12, no. 1, 2017.

[43]. Sabary O, Orlev Y, Shafir R, Anavy L, Yaakobi E, and Yakhini Z, “SOLQC: Synthetic Oligo Library Quality Control Tool,” BioRxiv, p. 840231, 2019.

[44]. Chandak S, Tatwawadi K, Lau B, Mardia J, Kubit M, Neu J, Griffin P, Wootters M, Weissman T, and Ji H, “Improved read/write cost tradeoff in DNA-based data storage using LDPC codes,” in 2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton). IEEE, 2019, pp. 147–156.

[45]. Lopez R, Chen Y-J, Ang SD, Yekhanin S, Makarychev K, Racz MZ, Seelig G, Strauss K, and Ceze L, “Dna assembly for nanopore data storage readout,” Nature Communications, vol. 10, no. 1, pp. 1–9, 2019.

[46]. Rashtchian C, Makarychev K, Rácz M, Ang SD, Jevdjic D, Yekhanin S, Ceze L, and Strauss K, “Clustering Billions of Reads for DNA Data Storage,” in Advances in Neural Information Processing Systems, 2017, pp. 3360–3371.

[47]. Ganguly S, Mossel E, and Rácz MZ, “Sequence Assembly from Corrupted Shotgun Reads,” in 2016 IEEE International Symposium on Information Theory (ISIT), 2016, pp. 265–269.

[48]. Navarro G, “A guided tour to approximate string matching,” ACM computing surveys (CSUR), vol. 33, no. 1, pp. 31–88, 2001.

[49]. Schimd M and Bilardi G, “Bounds and Estimates on the Average Edit Distance,” in International Symposium on String Processing and Information Retrieval. Springer, 2019, pp. 91–106.

[50]. Newman S, Stephenson AP, Willsey M, Nguyen BH, Takahashi CN, Strauss K, and Ceze L, “High density DNA data storage library via dehydration with digital microfluidic retrieval,” Nature Communications, vol. 10, no. 1, pp. 1–6, 2019.

[51]. Willsey M, Stephenson AP, Takahashi C, Vaid P, Nguyen BH, Piszczek M, Betts C, Newman S, Joshi S, Strauss K et al., “Puddle: A dynamic, error-correcting, full-stack microfluidics platform,” in Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2019, pp. 183–197.

[52]. Anavy L, Vaknin I, Atar O, Amit R, and Yakhini Z, “Data storage in DNA with fewer synthesis cycles using composite DNA letters,” Nature Biotechnology, vol. 37, no. 10, pp. 1229–1236, 2019.

[53]. Tabatabaei SK, Wang B, Athreya NBM, Enghiad B, Hernandez AG, Fields CJ, Leburton J-P, Soloveichik D, Zhao H, and Milenkovic O, “DNA punch cards for storing data on native DNA sequences via enzymatic nicking,” Nature Communications, vol. 11, no. 1, pp. 1–10, 2020.

[54]. Dubé D, Song W, and Cai K, “DNA Codes with Run-Length Limitation and Knuth-Like Balancing of the GC Contents,” in Symposium on Information Theory and its Applications (SITA), Japan, 2019.

[55]. Fei P and Wang Z, “LDPC Codes for Portable DNA Storage,” in 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019, pp. 76–80.

[56]. Gabrys R, Pattabiraman S, and Milenkovic O, “Mass error-correction codes for polymer-based data storage,” 2020 IEEE International Symposium on Information Theory (ISIT), pp. 25–30, 2020.

[57]. Jain S, Farnoud F, Schwartz M, and Bruck J, “Coding for optimized writing rate in dna storage,” 2020 IEEE International Symposium on Information Theory (ISIT), pp. 711–716, 2020.

[58]. Immink KAS and Cai K, “Design of capacity-approaching constrained codes for DNA-based storage systems,” IEEE Communications Letters, vol. 22, no. 2, pp. 224–227, 2017.

[59]. Lenz A, Siegel PH, Wachter-Zeh A, and Yaakobi E, “Anchor-based correction of substitutions in indexed sets,” in 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019, pp. 757–761.

[60]. Lenz A, Liu Y, Rashtchian C, Siegel PH, Wachter-Zeh A, and Yaakobi E, “Coding for Efficient DNA Synthesis,” in 2020 IEEE International Symposium on Information Theory (ISIT). IEEE, 2020.

[61]. Lenz A, Siegel PH, Wachter-Zeh A, and Yaakobi E, “Coding over sets for dna storage,” in 2018 IEEE International Symposium on Information Theory (ISIT). IEEE, 2018, pp. 2411–2415.

[62]. Lenz A, Rashtchian C, Siegel PH, and Yaakobi E, “Covering Codes Using Insertions or Deletions,” in 2020 IEEE International Symposium on Information Theory (ISIT). IEEE, 2020.

[63]. Lenz A, Siegel PH, Wachter-Zeh A, and Yaakobi E, “An upper bound on the capacity of the DNA storage channel,” in 2019 IEEE Information Theory Workshop (ITW). IEEE, 2019, pp. 1–5.

[64]. Pattabiraman S, Gabrys R, and Milenkovic O, “Coding for Polymer-Based Data Storage,” arXiv preprint arXiv:2003.02121, 2020.

[65]. Sima J, Raviv N, and Bruck J, “On coding over sliced information,” in 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019, pp. 767–771.

[66]. Shinkar T, Yaakobi E, Lenz A, and Wachter-Zeh A, “Clustering-correcting codes,” in 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019, pp. 81–85.

[67]. Conde-Canencia L and Dolecek L, “Nanopore DNA Sequencing Channel Modeling,” in 2018 IEEE International Workshop on Signal Processing Systems (SiPS). IEEE, 2018, pp. 258–262.

[68]. Heckel R, Mikutis G, and Grass RN, “A characterization of the dna data storage channel,” Scientific reports, vol. 9, no. 1, pp. 1–12, 2019. [PubMed: 30626917]

[69]. Heckel R, Shomorony I, Ramchandran K, and David N, “Fundamental limits of DNA storage systems,” in 2017 IEEE International Symposium on Information Theory (ISIT). IEEE, 2017, pp. 3130–3134.

[70]. Organick L, Chen Y-J, Ang SD, Lopez R, Liu X, Strauss K, and Ceze L, “Probing the physical limits of reliable DNA data retrieval,” Nature Communications, vol. 11, no. 1, pp. 1–7, 2020.

[71]. Guindon S, Dufayard J-F, Lefort V, Anisimova M, Hordijk W, and Gascuel O, “New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of phyml 3.0,” Systematic biology, vol. 59, no. 3, pp. 307–321, 2010. [PubMed: 20525638]

[72]. Bailey-Wilson JE and Wilson AF, “Linkage analysis in the next-generation sequencing era,” Human heredity, vol. 72, no. 4, pp. 228–236, 2011. [PubMed: 22189465]

[73]. Hoehn KB, Lunter G, and Pybus OG, “A phylogenetic codon substitution model for antibody lineages,” Genetics, vol. 206, no. 1, pp. 417–427, 2017. [PubMed: 28315836]

[74]. Murugan A, Mora T, Walczak AM, and Callan CG, “Statistical inference of the generation probability of t-cell receptors from sequence repertoires,” Proceedings of the National Academy of Sciences, vol. 109, no. 40, pp. 16 161–16 166, 2012.

[75]. Ralph DK and Matsen IV FA, “Likelihood-based inference of b cell clonal families,” PLoS computational biology, vol. 12, no. 10, p. e1005086, 2016. [PubMed: 27749910]

[76]. Pan K, Long J, Sun H, Tobin GJ, Nara PL, and Deem MW, “Selective pressure to increase charge in immunodominant epitopes of the h3 hemagglutinin influenza protein,” Journal of molecular evolution, vol. 72, no. 1, pp. 90–103, 2011. [PubMed: 21086120]

[77]. Watabe T, Kishino H, de Oliveira Martins L, and Kitazoe Y, “A likelihood-based index of protein–protein binding affinities with application to influenza ha escape from antibodies,” Molecular biology and evolution, vol. 24, no. 8, pp. 1627–1638, 2007. [PubMed: 17478433]

[78]. Levenshtein VI, “Binary codes capable of correcting spurious insertions and deletions of ones,” Prob. Inf. Trans, vol. 1, no. 1, pp. 8–17, 1. 1965.

[79]. Wang L and Jiang T, “On the complexity of multiple sequence alignment,” Journal of computational biology, vol. 1, no. 4, pp. 337–348, 1994. [PubMed: 8790475]

[80]. Notredame C, “Recent evolutions of multiple sequence alignment algorithms,” PLoS Computational Biology, vol. 3, no. 8, 2007.

[81]. Thompson JD, Linard B, Lecompte O, and Poch O, “A comprehensive benchmark study of multiple sequence alignment methods: current challenges and future perspectives,” PloS One, vol. 6, no. 3, 2011.

[82]. Yermanos AD, Dounas AK, Stadler T, Oxenius A, and Reddy ST, “Tracing Antibody Repertoire Evolution by Systems Phylogeny,” Frontiers in Immunology, vol. 9, pp. 2149–2162, 2018. [PubMed: 30333820]

[83]. Hsiao Y-C, Shang Y, DiCara DM, Yee A, Lai J, Kim SH, Ellerman D, Corpuz R, Chen Y, Rajan S et al., “Immune Repertoire Mining for Rapid Affinity Optimization of Mouse Monoclonal Antibodies,” in MAbs, vol. 11, no. 4. Taylor & Francis, 2019, pp. 735–746. [PubMed: 30900945]

[84]. Boyd SD, Gaëta BA, Jackson KJ, Fire AZ, Marshall EL, Merker JD, Maniar JM, Zhang LN, Sahaf B, Jones CD et al., “Individual variation in the germline Ig gene repertoire inferred from variable region gene rearrangements,” The Journal of Immunology, vol. 184, no. 12, pp. 6986–6992, 2010. [PubMed: 20495067]

[85]. Gadala-Maria D, Yaari G, Uduman M, and Kleinstein SH, “Automated analysis of high-throughput B-cell sequencing data reveals a high frequency of novel immunoglobulin V gene segment alleles,” Proceedings of the National Academy of Sciences, vol. 112, no. 8, pp. E862–E870, 2015.

[86]. Corcoran MM, Phad GE, Bernat NV, Stahl-Hennig C, Sumida N, Persson MA, Martin M, and Hedestam GBK, “Production of individualized V gene databases reveals high levels of immunoglobulin genetic diversity,” Nature Communications, vol. 7, p. 13642, 2016.

[87]. Zhang W, Wang I, Wang C, Lin L, Chai X, Wu J, Bett AJ, Dhanasekaran G, Casimiro DR, Liu X et al., “IMPre: an accurate and efficient software for prediction of T-and B-cell receptor germline genes and alleles from rearranged repertoire data,” Frontiers in immunology, vol. 7, p. 457, 2016. [PubMed: 27867380]

[88]. Ralph DK and Matsen IV FA, “Consistency of VDJ rearrangement and substitution parameters enables accurate B cell receptor sequence annotation,” PLoS Computational Biology, vol. 12, no. 1, p. e1004409, 2016. [PubMed: 26751373]

[89]. Medvedev P, “Modeling biological problems in computer science: a case study in genome assembly,” Briefings in bioinformatics, vol. 20, no. 4, pp. 1376–1383, 2019. [PubMed: 29394324]

[90]. Gusfield D, Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge university press, 1997.

[91]. Mitzenmacher M, “A survey of results for deletion channels and related synchronization channels,” Probability Surveys, vol. 6, pp. 1–33, 2009.

[92]. Sabary O, Yaakobi E, and Yucovich A, “The Error Probability of Maximum-Likelihood Decoding over Two Deletion/Insertion Channels,” in 2020 IEEE International Symposium on Information Theory (ISIT). IEEE, 2020.

[93]. Srinivasavaradhan SR, Du M, Diggavi S, and Fragouli C, “Algorithms for reconstruction over single and multiple deletion channels,” arXiv preprint arXiv:2005.14388, 2020.

[94]. ——, “Symbolwise map for multiple deletion channels,” in 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019, pp. 181–185.

[95]. Cheraghchi M and Ribeiro J, “An overview of capacity results for synchronization channels,” IEEE Transactions on Information Theory, 2020.

[96]. Chase Z, “New upper bounds for trace reconstruction,” arXiv preprint arXiv:2009.03296, 2020.

[97]. Borwein P and Erdélyi T, “Littlewood-type problems on subarcs of the unit circle,” Indiana University mathematics journal, pp. 1323–1346, 1997.

[98]. Borwein P, Erdélyi T, and Kós G, “Littlewood-type problems on [0, 1],” Proceedings of the London Mathematical Society, vol. 79, no. 1, pp. 22–46, 1999.

[99]. Krishnamurthy A, Mazumdar A, McGregor A, and Pal S, “Algebraic and Analytic Approaches for Parameter Learning in Mixture Models,” in Algorithmic Learning Theory, 2020, pp. 468–489.

[100]. ——, “Sample Complexity of Learning Mixture of Sparse Linear Regressions,” in Advances in Neural Information Processing Systems, 2019, pp. 10 531–10 540.

[101]. Alon N and Spencer JH, The probabilistic method. John Wiley & Sons, 2004.

[102]. Arora S and Barak B, Computational complexity: a modern approach. Cambridge University Press, 2009.

[103]. Bollobás B, “Almost every graph has reconstruction number three,” Journal of Graph Theory, vol. 14, no. 1, pp. 1–4, 1990.

[104]. Przykucki M, Roberts A, and Scott A, “Shotgun reconstruction in the hypercube,” arXiv preprint arXiv:1907.07250, 2019.

[105]. Radcliffe AJ and Scott AD, “Reconstructing subsets of $\mathbb{Z}_n$,” Journal of Combinatorial Theory, Series A, vol. 83, no. 2, pp. 169–187, 1998.

[106]. Peres Y and Zhai A, “Average-case reconstruction for the deletion channel: subpolynomially many traces suffice,” in 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 2017, pp. 228–239.

[107]. Chen X, De A, Lee CH, Servedio RA, and Sinha S, “Polynomial-time trace reconstruction in the smoothed complexity model,” arXiv preprint arXiv:2008.12386, 2020.

[108]. Chrisnata J, Kiah HM, and Yaakobi E, “Optimal Reconstruction Codes for Deletion Channels,” arXiv preprint arXiv:2004.06032, 2020.

[109]. Haeupler B and Mitzenmacher M, “Repeated deletion channels,” in 2014 IEEE Information Theory Workshop (ITW 2014). IEEE, 2014, pp. 152–156.

[110]. Kiah HM, Nguyen TT, and Yaakobi E, “Coding for Sequence Reconstruction for Single Edits,” in 2020 IEEE International Symposium on Information Theory (ISIT). IEEE, 2020.

[111]. Ban F, Chen X, Freilich A, Servedio RA, and Sinha S, “Beyond trace reconstruction: Population recovery from the deletion channel,” in IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS), 2019.

[112]. Ban F, Chen X, Servedio RA, and Sinha S, “Efficient Average-Case Population Recovery in the Presence of Insertions and Deletions,” in Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019), ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 145, 2019, pp. 44:1–44:18.

[113]. Narayanan S, “Population Recovery from the Deletion Channel: Nearly Matching Trace Reconstruction Bounds,” arXiv preprint arXiv:2004.06828, 2020.

[114]. Moitra A and Saks M, “A polynomial time algorithm for lossy population recovery,” in IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 2013, pp. 110–116.

[115]. Wigderson A and Yehudayoff A, “Population recovery and partial identification,” in IEEE 53rd Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 2012, pp. 390–399.

[116]. Davies S, Racz MZ, and Rashtchian C, “Reconstructing trees from traces,” in Conference On Learning Theory (COLT), 2019.

[117]. Narayanan S and Ren M, “Circular Trace Reconstruction,” 2020.

[118]. He L, Karau P, and Tabard-Cossa V, “Fast capture and multiplexed detection of short multi-arm DNA stars in solid-state nanopores,” Nanoscale, vol. 11, no. 35, pp. 16 342–16 350, 2019.

[119]. Karau P and Tabard-Cossa V, “Capture and translocation characteristics of short branched dna labels in solid-state nanopores,” ACS Sensors, vol. 3, no. 7, pp. 1308–1315, 2018. [PubMed: 29874054]

[120]. Chen K, Kong J, Zhu J, Ermann N, Predki P, and Keyser U, “Digital Data Storage Using DNA Nanostructures and Solid-State Nanopores,” Nano letters, vol. 19, no. 2, pp. 1210–1215, 2019. [PubMed: 30585490]

[121]. Seeman NC, “Nanomaterials based on DNA,” Annual review of biochemistry, vol. 79, pp. 65–87, 2010.

[122]. Lilley DM, “Structures of helical junctions in nucleic acids,” Quarterly reviews of biophysics, vol. 33, no. 2, pp. 109–159, 2000. [PubMed: 11131562]

[123]. Edgar RC, “MUSCLE: a multiple sequence alignment method with reduced time and space complexity,” BMC bioinformatics, vol. 5, no. 1, p. 113, 2004. [PubMed: 15318951]

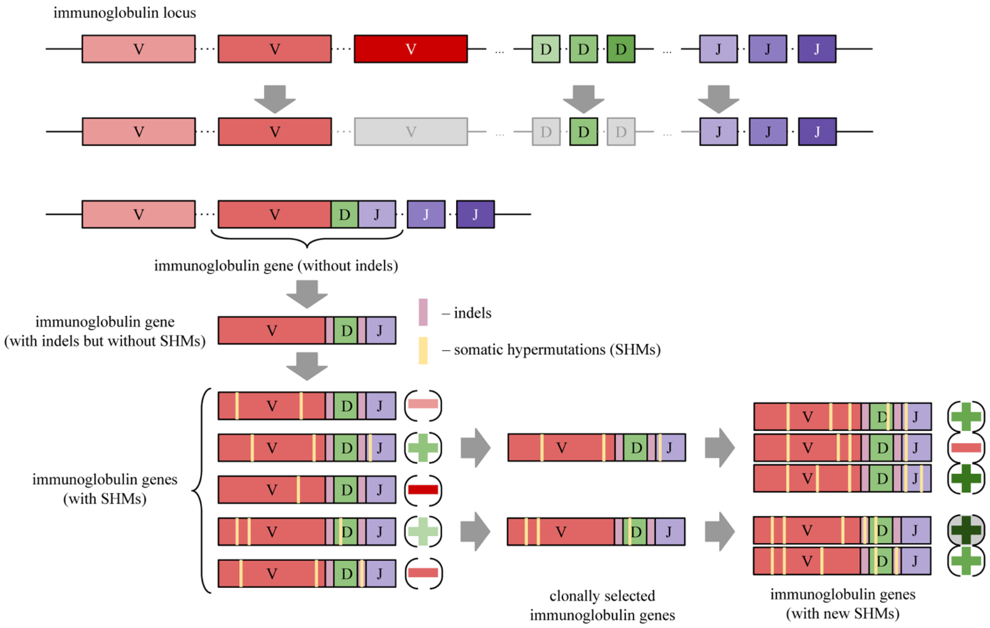


Fig. 1:

Generation of an antibody repertoire. The VDJ recombination affects the immunoglobulin locus that includes three sets of genes: V (variable), D (diversity), and J (joining). It randomly selects one gene from each set and concatenates them. The resulting sequence represents a potential immunoglobulin gene that encodes an antibody. However, this simple representation of an immunoglobulin gene is unrealistic since real immunoglobulin genes have indels at the V-D and D-J junctions. Somatic hypermutations (SHMs) further change the sequence of an immunoglobulin gene and thus affect its affinity. While some mutations increase affinity (sequences marked by the green '+' signs), other mutations reduce it (sequences marked by the red '−' signs). The clonal selection process iteratively retains antibodies with increased affinities and filters out antibodies with reduced affinities, thus launching an evolutionary process that eventually generates a high-affinity antibody able to neutralize an antigen (an antibody marked by a circled dark green '+' sign).

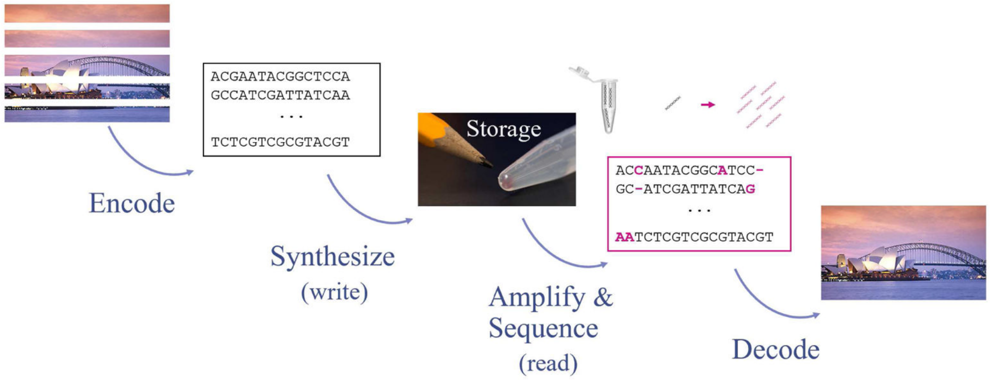


Fig. 2:

The DNA data storage and retrieval pipeline. Trace reconstruction problems come into play just before the Decode step.

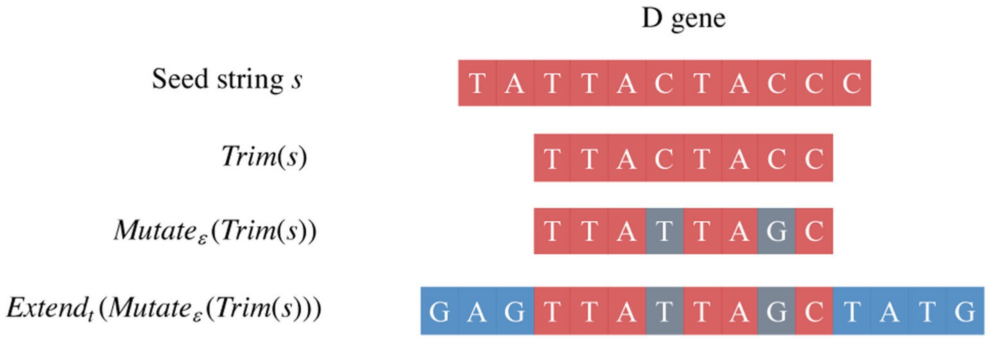
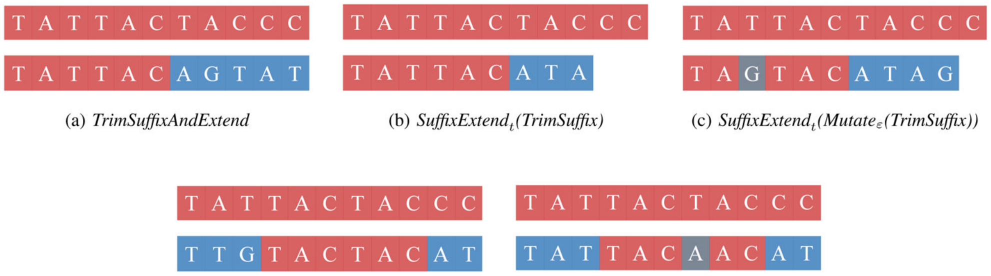

D gene
Seed string s
T A T T A C T A C C C
$Trim(s)$
T T A C T A C C
$Mutate_{\varepsilon}(Trim(s))$
T T A T T A G C
$Extend_{t}(Mutate_{\varepsilon}(Trim(s)))$
G A G T T A T T A G C T A T G

Fig. 3:

Trim, Mutate, and Extend operations model the process of generating a CDR3 of an immunoglobulin gene from a D gene using somatic hypermutations (shown in green) and random insertions (shown in blue).



(a) *TrimSuffixAndExtend*

(b) $SuffixExtend_t(TrimSuffix)$

(c) $SuffixExtend_t(Mutate_\varepsilon(TrimSuffix))$

(d) *TrimAndExtend*

(e) $Mutate_\varepsilon(TrimAndExtend))$

Fig. 4:

Trace generation for various trace reconstruction problems motivated by analysis of immunosequencing data. Insertions (i.e., random strings of random length) are shown in blue. Hypermutations are shown in green.

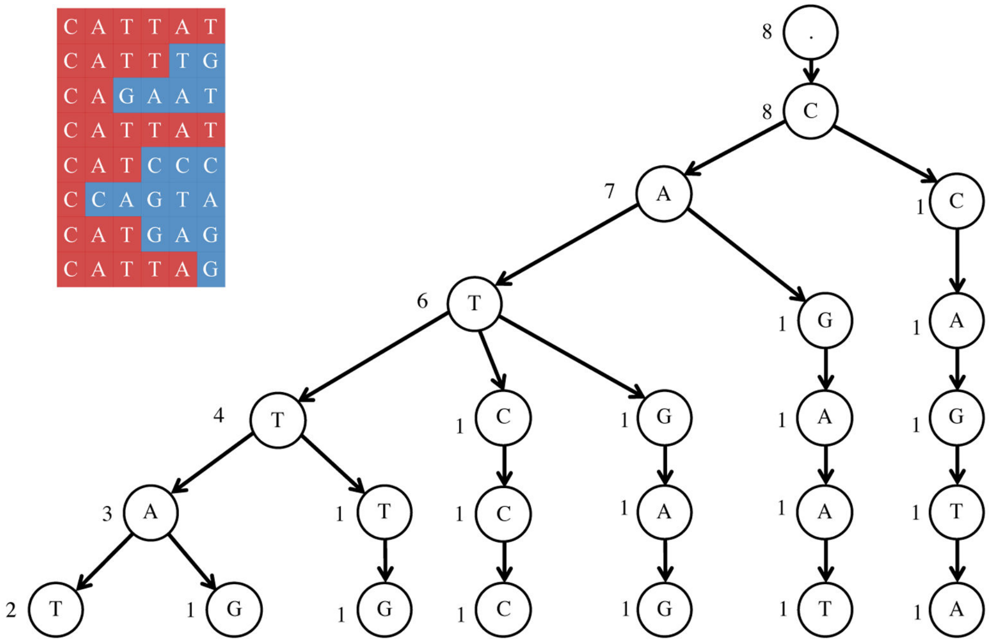
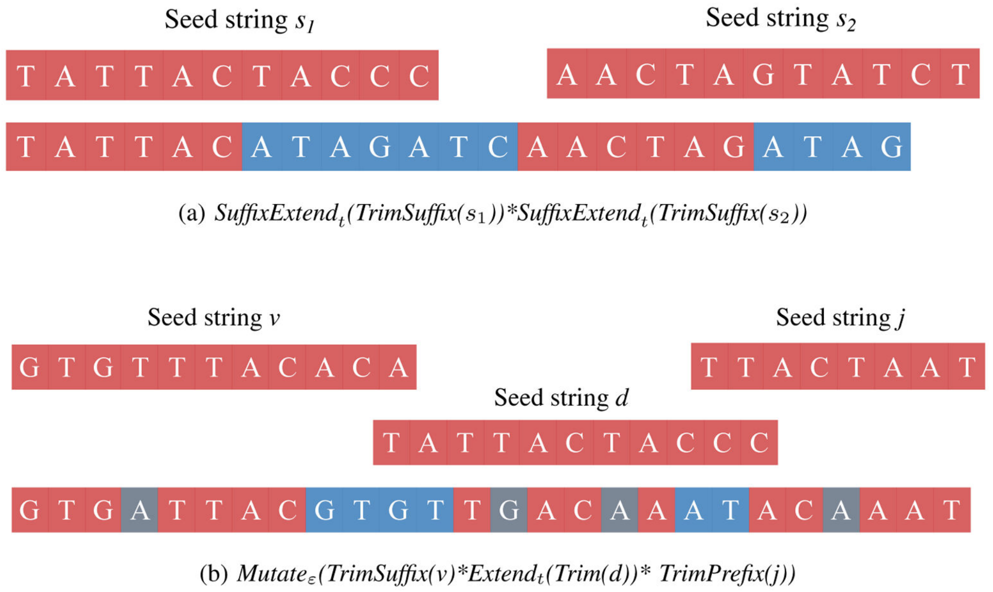


Fig. 5:

Illustration of the algorithm for solving the String Reconstruction Problem in the *TrimSuffixAndExtend* model. The set of traces is shown on the left, and their trie is shown on the right. The string associated with each vertex is the one that is formed by traversing the trie from the root node to the vertex. The values of $sim_t(C, s)$ for all vertices are shown.



(a) $SuffixExtend_t(TrimSuffix(s_1))*SuffixExtend_t(TrimSuffix(s_2))$

(b) $Mutate_\varepsilon(TrimSuffix(v)*Extend_t(Trim(d))* TrimPrefix(j))$

Fig. 6:

Trace generation that involves concatenation of multiple seed strings. Insertions are shown in light blue, hypermutations are shown in green. The most general model for the VDJ recombination is shown in (b).

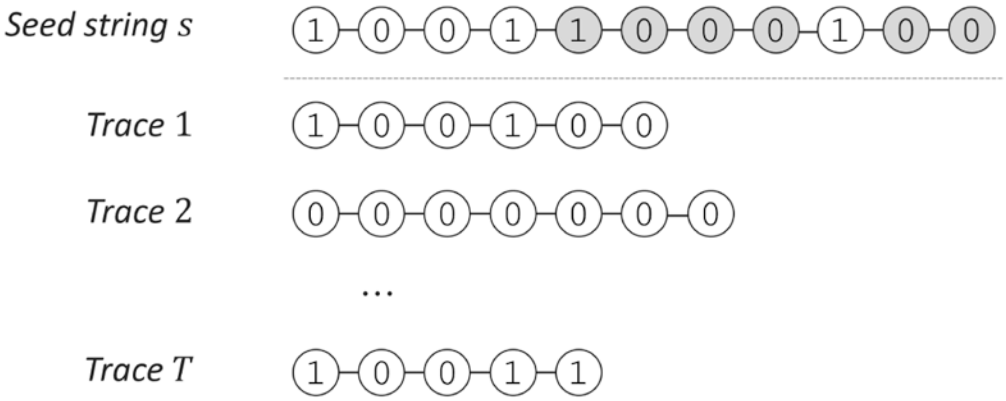


Fig. 7:

Seed string and example traces from the deletion channel. Gray circles indicate the deleted bits to generate the bottom trace.

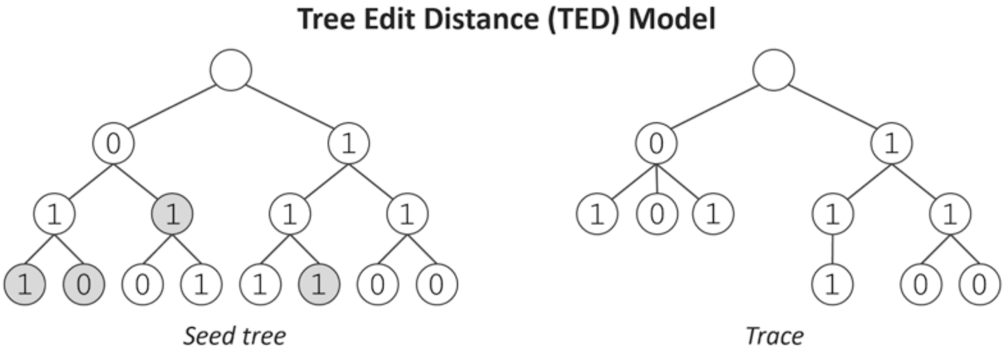


Fig. 8:
Labeled seed tree and example trace from the Tree Edit Distance deletion channel. Gray circles indicate deleted nodes.