Undergraduate Research Opportunity (UROP) Project Report

Simulating Noisy Channels in DNA Storage

by

Mayank Keoliya

Department of Computer Science
School of Computing
National University of Singapore

2021/2022

Simulating Noisy Channels in DNA Storage

by

Mayank Keoliya

Department of Computer Science
School of Computing
National University of Singapore

Project Number: U24307
Supervisor: Dr. Djordje Jevdjic

Deliverables:

Report: 1 Volume

Code: 1 Repository

2021/2022

Abstract

With increased interest and research in the use of synthetic DNA as a medium of archival storage, it has become important to quickly iterate and evaluate proposed approaches in storing DNA. However, current experiments on DNA have a high cost and high latency, leading to the need for cheap and fast simulation prior to experimentation. We propose and develop a simulator that approximates the noisy channel of DNA storage, which produces similar error profiles when compared to real experiments. Compared to existing simulators, our simulator converged closer to real data based on per-strand accuracy (15% v/s 38% difference) and per-character accuracy (1% v/s 6%) for the BMA algorithm; however, it did not adequately converge for the Iterative algorithm.

Further, we make the novel insight that the spatial distribution of errors within a strand is a key determinant of trace reconstruction accuracy; which is a factor that had not been considered by existing simulators. We conduct a sensitivity analysis on state-of-the-art trace reconstruction algorithms based on this insight, and use it show that the accuracy of the Iterative reconstruction algorithm can be significantly improved by performing two-way execution.

Keywords: DNA storage, error simulation, archival storage, trace reconstruction

Subject Descriptors:

E.2.1 Data Storage Representations

H.3.2 Information Storage

H.2.4 Data Management Systems

Contents

Chapter 1

Introduction

With the exponential increase in the production of data and stagnating costs of conventional electronic or magnetic storage, it has become essential to explore unconventional storage mediums with low cost, and high density and durability. The rapid obsolescence of storage technologies (espoused by the decline in use of floppy disks [1]) also poses challenges for archival storage which deals with storage over hundreds of years. Storing data in the form of synthesized DNA molecules (hereinafter referred to as *DNA storage*), has emerged as a promising means of dense and durable archival storage.

This paper is organised as follows. Chapter 1 provides an overview of the steps in DNA storage with an emphasis on the introduction of noise (errors) at each step. Chapter 2 reviews existing work on modelling and simulating errors in DNA storage, and shows the inadequacy of existing simulators to derive a problem statement. Chapter 3 proposes evaluation criteria for DNA simulators, and progressively considers various parameters to derive a better simulation model. We show that spatial distribution of errors is a key parameter in simulating error profiles, and then perform a sensitivity analysis to gauge the impact of spatial distribution on the trace reconstruction phase of the storage pipeline. Chapter 4 concludes with a discussion of the limitations and recommendations for future work.

This chapter introduces the mechanism and errors associated with the DNA storage pipeline. We then review the noise profiles of existing DNA technologies, which are fundamentally different in mechanism and thus in noise distribution.

1.1 Overview of DNA Storage

DNA storage allows for write-store-read operations on digital information. Writes, also called *synthesis*, produce physical DNA molecules of short length, called *strands*. Storing the strands simply requires their placement in a physical container which contains other DNA strands. Reads, also called *sequencing*, produce digital representations of DNA sequences on processing the corresponding DNA strands.

There are 4 types of molecules (also called *letters* or *bases*) in DNA *viz.* A, G, C and T. Strands of these molecules can be constructed with a desired sequence and length; examples include strands such as $GCTA$ (length = 4) and $AATCAG$ (length = 5). To convert from digital information (in

the form of bits) and DNA storage (in the form of strands) and vice-versa, a suitable encoding scheme can be devised. A trivial example of such an encoding would be A: *00*, G: *01*, C: *10*, and T: *11*. Illustratively, our earlier example $GCTA$ corresponds to *01 10 11 00*, and this conversion is performed during reads.

Writes are thus modelled as a transformation from a binary alphabet to a set of strings of fixed length L over the alphabet $\Sigma = \{A, G, C, T\}$ and vice-versa for reads. However, both write and read operations are **noisy transformations** which introduce errors stochastically. The introduction of errors is modelled as a noisy channel $(\Sigma_L)^N \rightarrow (\Sigma^*)^M$, where N strands of length L can suffer insertion, deletion or substitution of substrings, or even full deletion of the strand, to result in M reads of varying lengths, i.e not necessarily equal to L. Typically, $M \geq N$, i.e. the number of reads is more than the number of writes since the sequencing process magnifies the number of reads (see §1.1.1 below).

The ratio $M : N$, i.e. the ratio of the number of strands that are synthesized (references) to the number of copies that are sequenced is termed the *sequencing coverage*. A higher sequencing coverage leads to improved error correction, since more copies are available for every synthesized strand.

In sum, the aim of DNA storage is to:

1. Accept a file structure as a sequence of bits *viz.* a binary string
2. **Encode** the binary alphabet to DNA sequences *viz* an $AGCT$ string
3. Synthesize and store the DNA sequence (now called a *strand*)
4. **Retrieve** data according to queries
5. Sequence, **cluster** and **reconstruct** the DNA strands
6. Decode and correct errors to re-arrange the file structure

See Fig. 1.1 for a graphical overview.

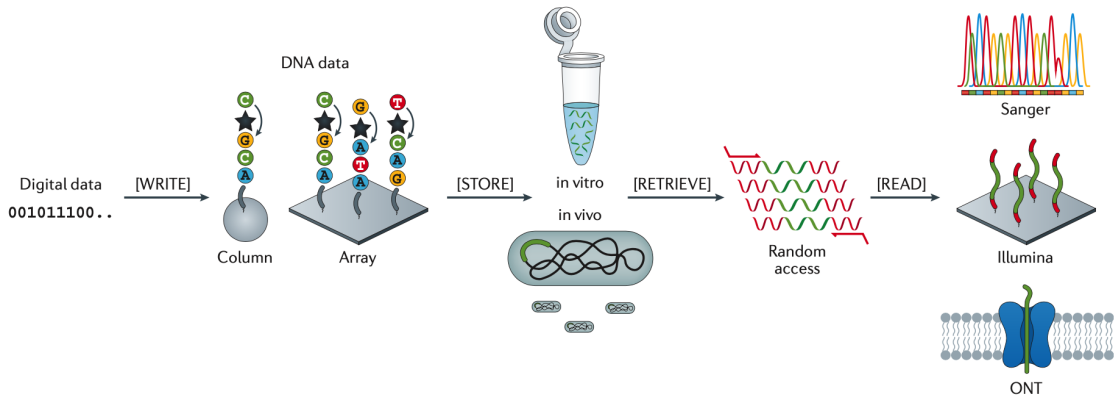


Figure 1.1: Overview of the major steps of digital data storage in DNA [5].

To reduce the probability of errors, two forms of redundancy are employed: *physical redundancy* and *logical redundancy*. Physical redundancy involves the generation of multiple copies of the same strand, from which the original strand is recovered in a process known as *trace reconstruction*.

Logical redundancy involves the addition of redundant information that can be used by an error-correction code to detect and correct errors during decoding.

Since redundancy comes with a high cost of synthesis, a key aim is to ensure that the logical density per letter (i.e the number of bits represented per DNA molecule) is maximized. The theoretical maximum, demonstrated by [13] is 2 bits per nucleotide, assuming zero redundancy. The physical density of DNA (i.e the number of molecules per unit weight) is another parameter to be maximized; fewer, longer strands are more dense than several, shorter strands.

1.1.1 Retrieval

Unlike conventional electronic and magnetic storage, DNA storage is not physically organized, making random-access and searching a non-trivial problem. The process that enables random-access viz. polymerase-chain reactions, also injects errors into the channel.

Initial studies ([9], [11]) sequenced and read all of the data in the DNA pool, without any scope for random access. This requires an *O(n)* search to read a particular chunk of data, and was tolerable due to the low amount of data encoded (650 KB and 630KB). However, this solution is not scalable, especially due to the extremely high potential density of 17 exabytes per gram of DNA.

Two studies by Yazdi et al. [25] and Bornholt et al. [4] independently solved the problem of random-access by the use of primers and polymerase-chain reaction (PCR). The DNA storage system was modelled as a key-value store, with the key being analogous to a filename, and the value being the file's contents. Each key was mapped to a unique sequence of 20 bases called a primer, and this primer was appended to each strand that encoded the file content. These primers allow the PCR to selectively amplify only those strands with a chosen primer sequence. However, the amplification is imperfect; strands of undesired files might remain, and even strands of desired files might be corrupted via substitution.

1.1.2 Clustering and Reconstruction

The output of sequencing is an unordered list of noisy copies of original strands i.e. $(\Sigma^*)^M$. Similar strands are clustered together based on the heuristic of edit distance, assuming that similar reads correspond to noisy copies of the same strand. However, clustering might itself be imperfect, since a noisy copy n' of a strand n might be clustered together with copies of another strand m. Under this setup, a sequence n (the designed DNA strand) is transmitted m times over the deletion-insertion-substitution channel and generates m noisy copies (the cluster). A DNA reconstruction algorithm is a mapping which receives the m noisy copies as an input and produces $\hat{n}$, an estimation of n, and the target is to minimize the distance between n and $\hat{n}$

Classic trace reconstruction algorithms can then be applied, including Multiple Sequence Alignment [24], Bitwise Majority Alignment (BMA) [3], Divider BMA [21] and others. All of these algorithms require consensus or majority voting for each position of the DNA strand, and thus are subject to deteriorating accuracy with lower sequencing coverage.

Table 1.1: Comparison of DNA sequencing technologies

Sequencing technology	1^{st} Gen. (Sanger)	2^{nd} Gen. (Illumina)	3^{rd} Gen. (Nanopore)
Cost (per Kb)	\$1-2	$\$10^{-5} - 10^{-3}$	$\$10^{-4} - 10^{-3}$
Error rate	0.001-0.01%	0.1-1%	10%
Sequencing length	500bp	25-150bp	10^5bp
Read speed (per Kb)	10^{-1} h	10^{-7}-10^{-4} h	10^{-7}-10^{-6} h

1.1.3 Decoding

The output of reconstruction is a (smaller) set of unordered, noisy, DNA sequences *in silico*, which are then decoded into ordered binary sequences within files.

DNA strands can suffer from two types of errors: *erasures* and *corruption*. Erasures occur due to failed PCR amplification, low sequencing coverage, or imperfect clustering where all noisy copies of a strand are classified as belonging to another strand. Erasures are detected easily when a strand is not present after reconstruction, for example, by detecting that a given location in a file is missing. Both physical and logical redundancy must be employed to correct them. Corruption occurs when a strand undergoes substitution during synthesis, sequencing or incorrect trace reconstruction. Several schemes, such as parity bit checks [11], low-density parity check code (LDPC) [6], generation of redundant XOR copies [4], Reed-Solomon codes [12], and fountain codes [10] have been employed to correct errors and ensure the original binary sequence is returned to the user.

1.2 Noise Profiles of DNA Technologies

As we have seen, the three critical steps where DNA molecules are altered undesirably are synthesis, PCR, and sequencing. The most widely synthesis technologies are Twist Bioscience, CustomArray and Integrated DNA Technology (IDT) [5]. All of the synthesis methods have two common properties. First, the synthesis of long strands (greater than 200 bases) leads to increasing error rates, especially at terminal positions. This is why short strands of length in the range 100-400 are used. Secondly, a 50% GC-ratio[1], since extreme GC-ratios have been found to be unstable. Strands with high GC-ratios are attracted to themselves in a self-loop, and form secondary structures that prevent accurate sequencing [28].

However, several studies ([4], [11], [10], [24]) have shown that most errors occur during sequencing. In general, sequencing is vulnerable to*homopolymers* which are repeated sequences of the same base such as $AAAAA$, and several encoding techniques have been employed to prevent their occurrence [11]. For low-error sequencing (such as Sanger and Illumina), corruption is the dominant concern, while for high-error sequencing (such as Nanopore), erasures are prevalent. A comparison of the sequencing technologies is provided in Table 1.1. The maximum length of a strand that can be successfully sequenced, termed the *sequencing length*, differs greatly as well. Further, Nanopore sequencing is particularly sensitive to *burst errors*, where 5 or more consecutive bases are either substituted or deleted [17].

[1] GC-ratio $= \frac{\text{No. of G and C bases}}{\text{Total no. of bases}} \times 100$

Since DNA is intended for archival storage over hundreds of years, it is important to ensure that reconstruction algorithms and error-correction schemes are robust not only when used with current sequencing technologies, but also future sequencing technologies that might have different error profiles. As evident from Table 1.1, trends in sequencing technologies suggest that a higher throughput (and lower latency) technology tends to be associated with higher error rates. This acts as another motivator for a DNA simulator, such that a user can be guaranteed a certain degree of success in retrieval of information regardless of future sequencing technologies.

Chapter 2

Literature Review

This chapter surveys the current state of work on DNA storage simulation, indicates areas for improvement in the same, and defines the problem statement for the research project.

2.1 Survey of Prior Work

The problem of accurately modeling the noisy channels in DNA storage is key for designing simulators for DNA storage, as well as for devising highly accurate error-correction schemes. In [14], a rudimentary model is proposed, where data is written on M molecules, PCR amplification is assumed to be uniform, and sequencing is modelled as drawing N times uniformly at random with replacement from the M DNA strands. However, the study acknowledges that the proposed model only accounts for erasures (deletions of strands), since every read and write is assumed to be error-free. A Poisson distribution, instead of a uniform distribution, is suggested for PCR amplification.

Heckel et al. [13] improved on this model, and characterizes errors as a Insertion-Deletion-Substitution (IDS) channel. This study is the first to describe the DNA storage system as a noisy channel with a multiset of M DNA strands of length L as input, and the output as sampling N times independently from the multiset, and then disturbing the sampled strands with insertions, deletions and substitutions. The following types of errors are posited:

1. Strands might not be successfully synthesized, and some might be synthesized more times than others (i.e higher number of noisy copies).
2. During storage, DNA strands might decay, or be lost.
3. Sequencing only draws a fraction of molecules. IDS errors might occur, with higher probability of errors in homopolymer subsequences and strands with unbalanced GC-ratios.

Heckel et al. analyze experimental data to show that PCR has a preference for some sequences over others, which distorts the copy number distribution of individual strands. It demonstrates that synthesis and sequencing errors are dominated by deletions and substitutions respectively. Conditional error probabilities for mistaking a certain base for another were computed, which showed that, for example, mistaking T for C or A for G was much more likely ($p \sim 0.4$) compared to other

combinations ($p \sim 0.01$). The distribution of the number of reads per synthesized strand (using Illumina) was found to be approximately negative binomial distributed, unlike prior assumptions of a uniform distribution or even a constant coverage.

Other related work in the field of simulating DNA is only partially applicable to its use in digital storage. For example, `MESA` [22] simulates DNA synthesis, sequencing, and, notably, PCR amplification as well. It also includes a module that accounts for storage decay, as well as including adjustments for homopolymers, GC-ratios and conditional substitution probabilities that are absent from `DNASimulator`. However, the datasets used to compute the error statistics are based on DNA stored *in vivo*, i.e. in living organisms such as *E. coli*. Further, only one noisy copy is generated for a given strand, which prevents simulation of clusters and reconstruction.

Other works have partially addressed specific steps such as sequencing, albeit in the context of bioinformatics. Two studies, ([23], [16]) have attempted to simulate errors in sequencing genomic DNA fragments using Nanopore, while there has been work on Illumina sequencing as well [26]. However, these studies have limited applicability for this research project.

2.2 Study of Existing Simulators

Currently, only one prior work has attempted to devise an end-to-end simulator for the purposes of DNA storage *viz.* `DNASimulator` by Gadihh et al. [7]. The simulator uses a dictionary of pre-computed error statistics, generates noisy copies given an input file of strands, and provides a suite of reconstruction algorithms to test the simulated noisy copies.

2.2.1 Algorithm

The algorithm used to inject errors in `DNASimulator` is given below. In the algorithm, a unique dictionary E is predetermined for each pair of synthesis and sequencing technology, and the number of noisy copies N is tunable by the user. The combination of these two parameters aims to reduce the different error injections in synthesis and into a single-pass injection. The errors introduced at different stages are not modelled separately, and the error-dictionary is computed by summarizing experimental results using the different technologies. Errors introduced due to PCR amplification or due to low sequencing coverage are not modelled explicitly. Furthermore, only 4×4 types of errors are considered i.e. insertion, deletion, substitution and long-deletion for the different bases.

2.2.2 Evaluation

To test the fidelity of simulation of `DNASimuator`, we propose per-strand and per-character accuracy as criteria for evaluation for simulators (see §3.1 for a broader discussion of suitable criteria). Data generated by an ideal simulator should have the same per-strand and per-character accuracy as real data when reconstructed with different trace reconstruction algorithms.

Algorithm 1 Generating noisy copies for reference strands with `DNASimulator`

$E \leftarrow P_{A,del}, P_{A,subs}, P_{A,ins}, ..., P_{T,ins}$ ▷ `Base-wise probabilities for error types`

$N \leftarrow ...$ ▷ `Desired coverage for strand`

for *strand in reference-strands* **do**
  for $k \leftarrow 1$ *to* N **do**
    $copy_k \leftarrow \emptyset$
    for *base in strand* **do**
      prob $\leftarrow$ `random()` ▷ `random real number` $\in (0, 1)$
      if $prob \leq E_{base,subs}$ **then**
        $b \leftarrow$ random base $\in \{A, G, C, T\}$
        $copy_k \leftarrow copy_k + b$
      end
      if $prob \leq E_{base,subs} + E_{base,ins}$ **then**
        $b \leftarrow$ random base $\in \{A, G, C, T\}$
        $copy_k \leftarrow copy_k + base + b$
      end
      if $prob \leq E_{base,subs} + E_{base,ins} + E_{base,del}$ **then**
        continue
      end
      else
        $copy_k \leftarrow copy_k + base$
      end
    end
  end
end

We provide an analysis of trace reconstruction accuracies to demonstrate the deficiencies in `DNASimulator` when applied to high-error regimes such as Nanopore. In addition to `DNASimulator`, we design a naive simulator [15] that ignores (i) conditional base-wise probabilities, and (ii) long-deletions.

A Nanopore dataset from [3] was used, which contained 10,000 clusters, 269, 701 noisy copies, and an average coverage of 26. The coverage of the Nanopore clusters ranged from 0 - 164, with an aggregate error of 5.9% as reported in [3].

We prepared two simulated datasets, one each using `DNASimulator` and our naive simulator, where each cluster has a **custom coverage** equal to the coverage in the real Nanopore data. An additional dataset with a fixed coverage of 26 was also generated using `DNASimulator`. We ran various trace reconstruction (TR) algorithms on these four datasets (see Table 2.1) to get preliminary evidence on the efficacy of `DNASimulator`.

We observed that the per-strand accuracy of simulated data was consistently *greater* than the per-strand accuracy of real data. Further, `DNASimulator` performs roughly the same as a naive simulator. This serves as preliminary evidence showing the deficiency of `DNASimulator`.

However, the prior analysis in Table 2.1 does not control for coverage, which can serve as a confounding factor in the per-strand accuracy of different TR algorithms. Particularly, it is

Table 2.1: Comparison of Per-Strand Accuracy of TR Algorithms on Real and Simulated Data

Data	Coverage	BMA (%)	DivBMA (%)	Iterative (%)
Real Nanopore	Custom	77.88	2.73	83.16
Naive Simulator	Custom	93.77	3.33	100
`DNASimulator`	Custom	95.91	0.38	99.1
`DNASimulator`	26	94.12	0.07	100.

Table 2.2: Comparison of Accuracy of TR Algorithms at Fixed Coverage

Data	Coverage	BMA		Iterative	
		Per-Strand (%)	Per-Char (%)	Per-Strand (%)	Per-Char (%)
Nanopore	5	29.04	87.74	66.70	90.32
DNASimulator	5	68.21	93.45	90.60	99.31
Nanopore	6	36.88	89.26	78.88	94.48
DNASimulator	6	81.09	95.55	98.04	99.87

possible that clusters with higher coverages have lower error, which will not be captured by either the naive simulator or `DNASimulator`. Thus, we ran additional experiments using a fixed coverage for both real data and simulated data (Table 2.2).

After controlling for coverage, both per-strand and per-character accuracy of simulated datasets continued to be greater than those for real data. This demonstrates that static error-profiling as used in `DNASimulator` is not adequate for simulating DNA storage.

2.2.3 Discussion

In this subsection, we posit reasons for the inadequacy of `DNASimulator`. The error model of `DNASimulator` is localized to a single base, and ignores several characteristic sources of errors are analyzed by Heckel et al. [13]. For example, all errors are assumed to be independent of a base's position in a strand, despite studies illustrating that error rates increase exponentially at terminal positions [5], [17]. The increased error rate due to presence of homopolymers as discussed earlier is also not included. The sequencing coverage of all strands is assumed to be uniform, which is contrary to empirical data showing that sequencing coverage assumes a normal distribution across all strands [4]. Entropy due to decay during storage is not considered. The conditional probabilities of substitutions are modelled as roughly equal, since a random base $\in \{A, G, C, T\}$ is chosen uniformly; this is contrary to observations specified above. Finally, `DNASimulator` fails to account for the possibility errors due to strand-strand interactions, since the injection of errors for every strand is performed independently.

Additionally, `DNASimulator` has recently been used as a synthetic data generator (SDG) to

train `DNAformer`, a neural network that reconstructs strands from imperfect clusters [2]. The study justifies the use of `DNASimulator` by demonstrating that the proposed neural network trained on real data performs worse than one trained on data generated by the SDG. Deep learning approaches have been applied partially in simulating Illumina sequencing as well [27]. A simulator superior to `DNASimulator` could instead be used to train these neural networks, which would lead to improved reconstruction accuracy.

2.3 Problem Definition

With the notable exception of `DNASimulator` [7], most of the existing work on simulating DNA has dealt with genomics DNA, and with specific steps of the DNA sequencing and synthesis pipeline. `DNASimulator` is the only end-to-end error simulator that aims to approximate the noisy channels of DNA storage. However, it uses a simple model that treats errors as independent of each other and of the position within a strand, does not distinguish between errors during synthesis, PCR or sequencing, and does not offer the ability to simulate different sequencing coverages, which are key in testing reconstruction algorithms and error-correction schemes.

This research project designs a simulator for DNA storage that accurately models the errors introduced at different positions within a DNA strand. The problem can be formalized as follows: Given a multiset $(\Sigma_L)^N$ over an alphabet $\Sigma = \{A, G, C, T\}$, a set of parameters including but not limited to $method_{synth}$, $method_{seq}$, $method_{PCR}$, $time_{storage}$..., simulate an IDS noisy channel $(\Sigma_L)^N \rightarrow (\Sigma^*)^M$ that generates another multiset $(\Sigma^*)^M$ of strings of varying length. The synthetically generated $(\Sigma^*)^M$ should be minimally distant from the reads from real wetlab experiments $(\Sigma^*_{real})^{M'}$, according to suitable metrics for measuring distance (see Chapter 3 for a discussion of possible metrics).

Additionally, the project uses a data-driven approach that does not require manual intervention and classification of key probabilities. It also provides a command-line interface based on `DNASimulator` to interact with the simulator (see appendix A for details).

Chapter 3

Simulation Model

This chapter outlines possible metrics to evaluate the accuracy of a DNA simulator, progressively refines this paper's simulator based on these metrics, and conducts a sensitivity analysis based on the spatial distribution of errors.

3.1 Evaluation Criteria

As discussed earlier, a DNA storage simulator generates an ordered list of M strings of the form $m_{11}, m_{12}, m_{13}, .., m_{21}, m_{22}, ...m_{n1}...m_{nk}$ where a string m_ij represents the jth noisy copy of a strand n_i. Several approaches can be considered to evaluate the generated M strings to determine the accuracy of the simulator and compare it to real data from wetlab experiments. However, prior to evaluation, we must choose whether the generated noisy copies are clustered imperfectly or perfectly. In imperfect clustering, the noisy copies are shuffled randomly to yield an unordered set M' that resembles an actual sequencing read-out in a wetlab experiment. The unordered set M' is then evaluated using the state-of-the-art clustering algorithms [18], [8]. However, this might lead to introduction of errors of a characteristic distribution due to the nature of the clustering algorithm. To mitigate this, we can use perfect clustering (also termed pseudo-clustering) where the ordered output of the simulator is considered to be already clustered.

Regardless of the choice of clustering, we now consider possible choices for metrics and evaluate their trade-offs:

1. **Error statistics**: Compute the error profiles of the generated noisy copies, for example by measuring error-rates per base, conditional probabilities of substitution $P(b_1|substitution_{b_2})$ for all pairs $b_1, b2$. The latter can be generalized to a frequency distribution for all possible types of errors of the form $x_1x_2x_3...x_ix_{i,1}x_{i,2}...x_{i,n}...x_l$ where x_i is the original base in a strand, and $x_{i,j}$ are the (possibly empty) replacements of it under the IDS channel. The χ^2 distance between the frequency histograms of the real data and the simulated data would be used to evaluate the simulator in this case.

2. **Normalized edit, Hamming or Levenshtein distance**: Compute the normalized edit,

Hamming or Levenshtein distance between clusters sourced from real data and simulated data.

3. **Gestalt Pattern Matching** : Compute the gestalt pattern matching score to compare the similarity of clusters sourced from real data and simulated data. Gestalt score [19] is used to compute the similarity of two strings. Given two strings, S_1 and S_2, we set K_m to be the number of matching characters in both strings. Matching characters are defined as the longest common substring (LCS) of S_1 and S_2 plus recursively the number of matching characters on either side of the LCS. The gestalt score is then computed as $D_{score} = \frac{2K_m}{|S_1|+|S_2|}$.

Figure 3.1: Example of gestalt matching on `WIKIMEDIA` and `WIKIMANIA` [5].

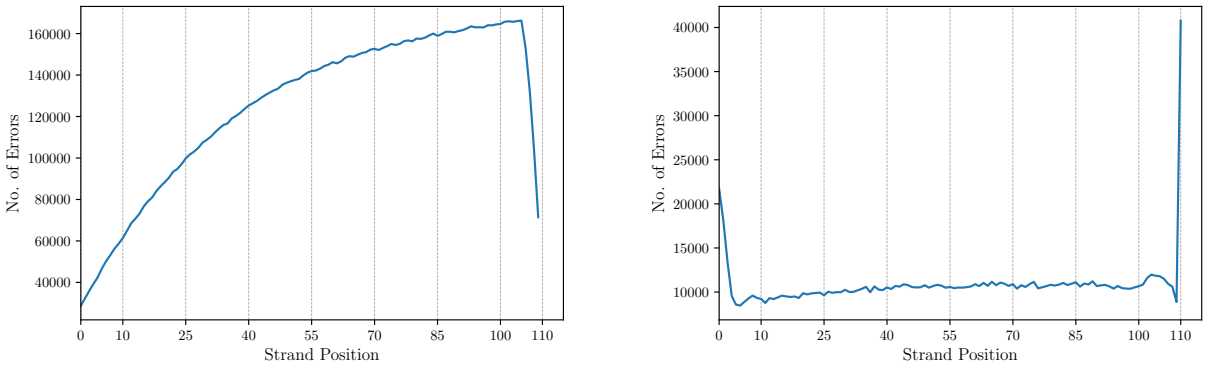
Importantly, gestalt pattern matching algorithm also generates the matching blocks in two strings as a by-product of the score computation. For example, in Fig. 3.1, the terms `WIKI` and `IA` are matching blocks in the strings `WIKIMEDIA` and `WIKIMANIA`. The substring `AN` is considered a substitution of `ED` in the first string. In context of DNA storage, the gestalt matching effectively generates the aligned (or matched) portions of a reference strand and a reconstructed strand; it corrects the misalignment due to the noisy channel.

4. **Accuracy of reconstruction algorithms**: The above methods attempt to compare the similarities between simulated and real data using closed-form analysis and probabilities. However, our motivation is not to generate DNA distributions that conform to a predetermined distribution but to ensure that simulated and real data are agnostic to the end-user of the reconstruction algorithms and the decoding, such that they may be optimized. Thus, we propose that a simulator should be evaluated on the basis of the differences in accuracies (per-strand and per-character) when simulated and real data are passed to a suite of reconstruction algorithms. Two algorithms are used in this report: BMA Look-Ahead [3], and Iterative Reconstruction [21]. *Per-strand accuracy* is defined as to the percentage of reference strands that are reconstructed without errors, and *per-character accuracy* is defined as the percentage of characters in the reference strands that are reconstructed without errors (i.e. with the correct base at the correct position).

Thus, we use per-strand and per-character accuracy after trace reconstruction as our key metric for evaluating the simulator. We use Hamming distance and gestalt pattern matching to visualize trace reconstruction outputs and to conduct sensitivity analyses. Both the Hamming and gestalt-aligned comparisons show the errors remaining *after* applying the reconstruction algorithms on noisy clusters.

3.2 Data

In this section, we choose and justify the sequencing dataset used, describe its error profile, and determine relevant sequencing coverages for our evaluation.



(a) Hamming errors

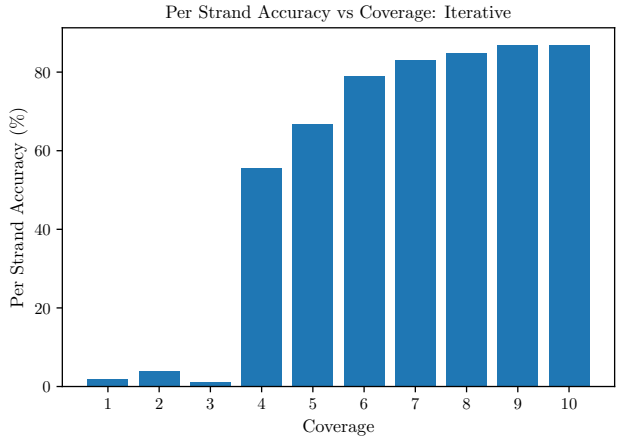
(b) Gestalt-aligned errors

Figure 3.2: Analysis of noise in Nanopore dataset *before* reconstruction

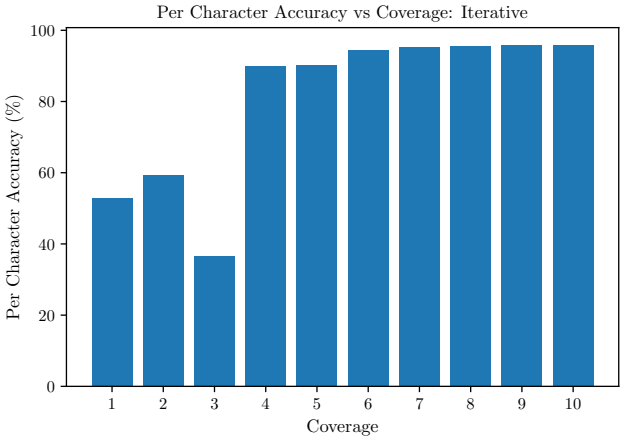
As in §2.2.2, we use the Nanopore dataset from [3], over Illumina datasets. This is because Nanopore sequencing has a much higher error rate (5%) than Illumina (0.1%), which allows us to distinguish between significant changes in reconstruction accuracies and random noise. The Nanopore dataset comprises 10,000 reference strands of length 110 each, and 269, 709 noisy copies, with an average coverage of $\frac{269709}{10000} = 26.97$. It includes 16 empty clusters or erasures, where no noisy copy could be recovered for the reference strand. We first ran Hamming and gestalt-aligned tests to determine the presence and sources of errors in the noisy copies. Hamming graphs indicate every presence of an error within a strand (Fig. 3.2a), while gestalt-aligned graphs indicate the *sources* of misalignment within a strand (Fig. 3.2b). For example, given a reference strand $r =$ `AGTC` and noisy copy $c =$ `ATC`, and 0-based indices, the Hamming error occurs at c_1, c_2 and c_3, however, the gestalt-aligned error occurs only at c_1 due to the deletion of `G`. The magnitude of gestalt-aligned errors is thus always lower than that of Hamming errors.

The Hamming comparison is expected, since any error at the beginning of a strand will propagate forward, leading to a linear trend upto position 110 (original strand length), after which there is a drop in errors, since the number of noisy copies with length greater than 110 is low. The gestalt-aligned comparison indicates that most errors occur at the *terminal positions* of the strand, which is predicted by [13]. However, end of the strand has almost twice the number of errors as the beginning of the strand.

In order to control for coverage, and choose relevant coverages for our analysis, we then ran the Iterative reconstruction algorithm on the Nanopore dataset at coverages ranging from 1 - 10. First, all clusters were shuffled. Only clusters with coverage greater than or equal to 10 were chosen, thus 1006 clusters were discarded at this step, since they did not have a minimum coverage of 10. For clusters with coverage $n \geq 10$, the remaining $n-10$ copies were discarded. The trace reconstruction for coverage i was measured by choosing the first i copies within each cluster. For example, for $N = 5$, the first 5 copies in each strand were chosen, and for $N = 6$, the first 6 copies were chosen. This ensured that the higher coverages differed from the lower coverage only in the extra copies chosen; the first few strands remained the same, and thus had the same error profile. Fig. 3.3 shows the per-strand and per-character accuracies for each coverage. Both the per-strand accuracy and per-character accuracy increase rapidly at coverages 4, 5 and 6, and then stabilize beyond



(a) Per-Strand Accuracy



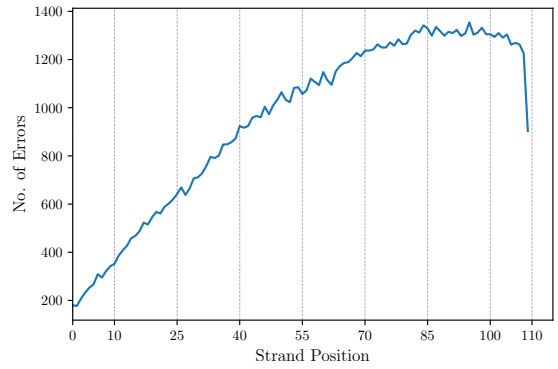
(b) Per-Character Accuracy

Figure 3.3: Accuracy of Iterative Reconstruction at $N \in 1...10$

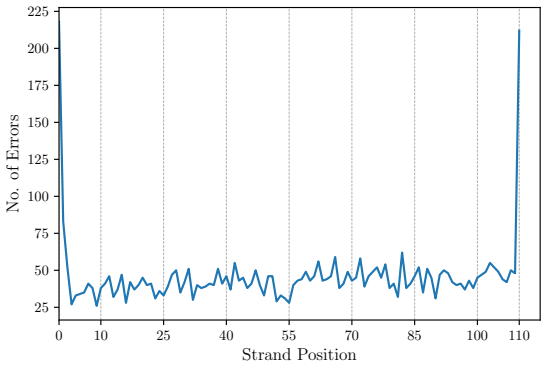
coverage = 7. Coverages of $N = 5$ and $N = 6$ are chosen as our reference metrics for per-strand and per-character accuracy due to the rapid fluctuation in that range.

After choosing suitable coverage points, we analyzed the outputs of running BMA and Iterative reconstruction algorithms at $N = 5$ and $N = 6$ on the Nanopore dataset. The comparisons for $N = 5$ are provided in Fig. 3.4. The corresponding comparisons for $N = 6$ yielded identical results, and are supplied in appendix C. The behaviour across both coverages is identical. The Hamming comparison for the Iterative algorithm is linear, as described earlier, due to the propagation of errors. The lack of symmetry points to a significant weakness in the Iterative algorithm mechanism, since the algorithm can potentially be improved by running it twice, on the strand and its reverse. This behaviour is discussed further in §3.4. The gestalt-aligned comparison of Iterative mirrors the error profile of Nanopore data (Fig. 3.2b), since errors occur at terminal positions.

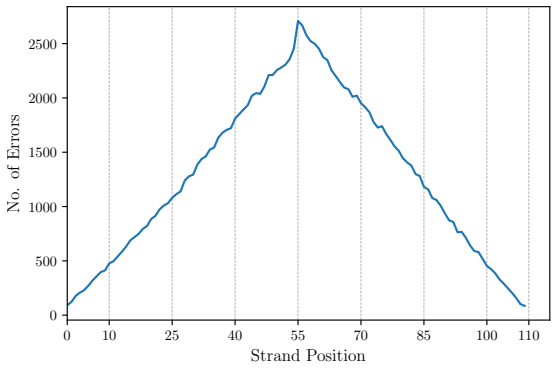
The Hamming comparison for BMA is symmetrically A-shaped, which is expected, since unlike Iterative, BMA performs a two-way execution on the cluster and its reverse as well. The first half of the forward execution is concatenated with the first half of the backward execution. Thus, errors propagate to the middle of the strand (position 55). The gestalt-aligned comparison reflects this, since the source of the misalignment is the middle position.



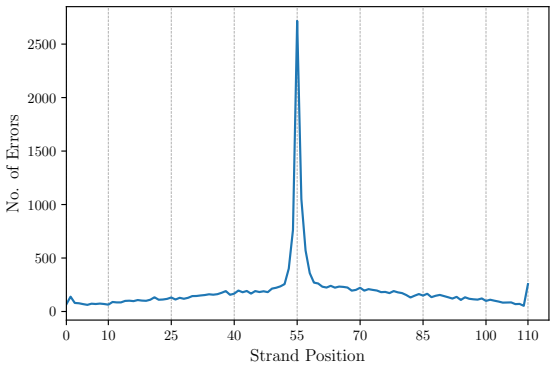
(a) Hamming errors for Iterative



(b) Gestalt-aligned errors for Iterative



(c) Hamming errors for BMA



(d) Gestalt-aligned errors for BMA

Figure 3.4: Post-reconstruction analysis of Nanopore data at $N = 5$

3.3 Factors Considered

A naive simulator only models three parameters, which are the aggregate probabilities of (i) insertion (ii) deletion and (iii) substitution. In this section, we progressively consider four additional parameters : the type of base (A, G, C or T), the proximity of errors, spatial distribution of errors, and second-order orders (described later). We show that the trace reconstruction accuracy of simulated data tends to that of real data for BMA, but not for the Iterative algorithm, due to its sensitivity to spatial distribution. Results are summarized in Tables 3.1 and 3.2.

Table 3.1: Comparison of Accuracy of TR Algorithms at $N = 5$

Data	BMA		Iterative	
	Per-Strand (%)	Per-Char (%)	Per-Strand (%)	Per-Char (%)
Nanopore	29.04	87.74	66.70	90.32
Naive Simulator	68.21	93.45	90.60	99.31
" + Cond. Prob + Del	59.65	91.39	92.20	99.35
" + Spatial Skew	47.86	89.49	35.36	82.15
" + 2^{nd}-order Errors	44.78	88.67	33.87	77.39

Table 3.2: Comparison of Accuracy of TR Algorithms at $N = 6$

Data	BMA		Iterative	
	Per-Strand (%)	Per-Char (%)	Per-Strand (%)	Per-Char (%)
Nanopore	36.88	89.26	78.88	94.48
Naive Simulator	81.09	95.55	98.04	99.87
" + Cond. Prob + Del	73.04	93.13	98.10	99.88
" + Spatial Skew	63.44	92.72	71.57	94.36
" + 2^{nd}-order Errors	58.19	91.50	69.41	91.34

3.3.1 Type of Base & Proximity of Errors

`DNASimulator` accounts for the type of base and the proximity of errors, but the values of the parameters are predetermined (hard-coded into the simulator). We provide an algorithm to compute these values.

As noted earlier, Heckel et al's model ([13] assumes that the occurrence of errors at a particular position is independent of the type of base (A, G, C or T) at that position. However, due to chemical properties of affinitive bonding, the A-T and G-C are likely to be substituted for each other. Thus, it is essential to use conditional probabilities for insertion, substitution and deletion given the occurrence of a base, e.g. $P(ins|A)$, $P(subs|G)$, and so on.

Likewise, it is assumed that the occurrence of errors at a particular position is is independent of the occurrence of errors at neighbouring positions. However, due to the nature of sequencing, it is likely that errors occur in groups [21], and in particular, in long deletions *viz.* consecutive deletions of length more than 2 [20].

Given a reference strand and its noisy cluster, we require the exact sequence of errors for each noisy copy to compute the parameters for conditional probabilities and long deletions. However, it is impossible to know which sequence of errors results in the noisy copies. For example, given a reference strand $r =$ `AGCG` and its noisy copy $c =$ `AGG`, then we have several sequences of operations

that are (unequally) possible: deletion of C at c_2, deletion of G at c_3 followed by substitution of C at c_2 with G, etc. We use the edit distance operations as a proxy to obtain the sequence of errors with the maximum likelihood (e.g. we choose the first case of deletion for the previous example). The algorithm is provided in Appendix B.

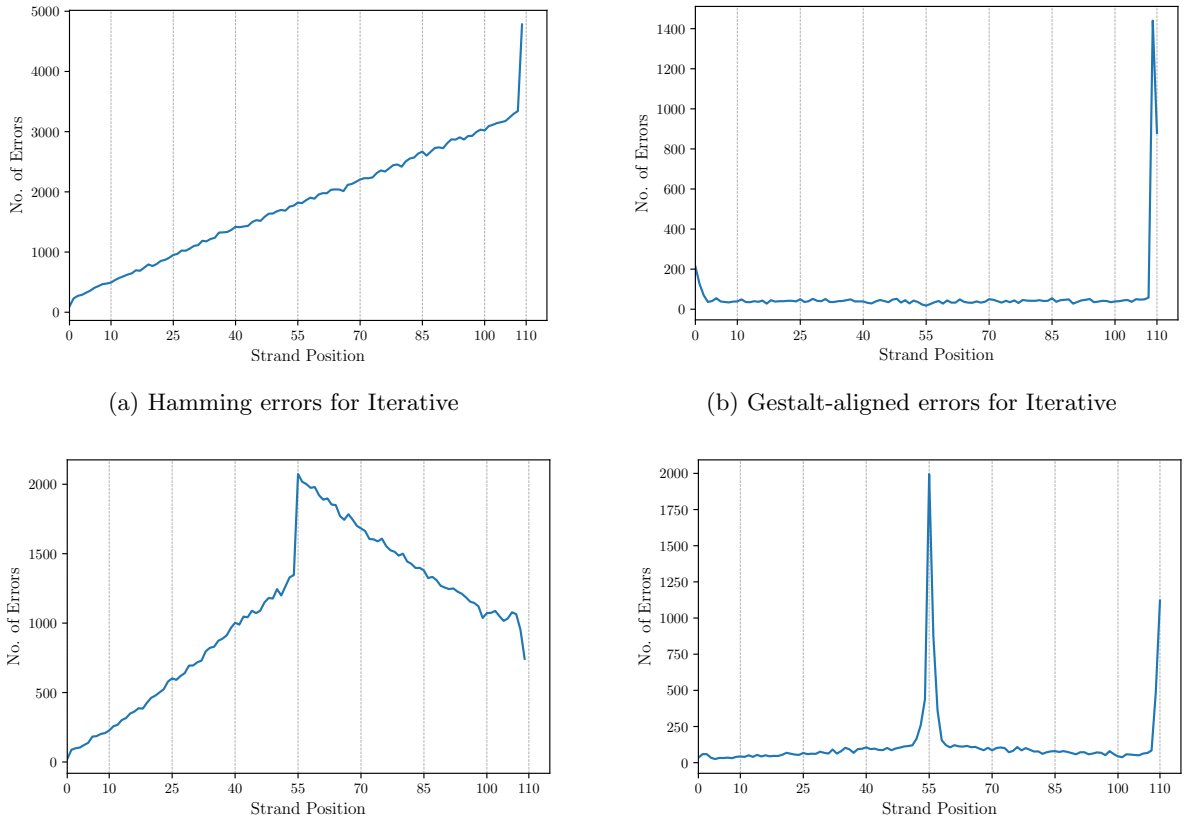
The edit distance operations are used to compute the conditional and long deletion probabilities. The long deletion probability, p_ld was 0.33%, with a mean length of 2.17. The ratios of the lengths of deletions were as follows: 2 (84%), 3 (13%), 4 (1.8%), 5 (0.2%), 6 (0.02%), $\geq$ 7 (0%). These parameters then used to simulate data, which improved on the naive simulator across all coverages (Tables 3.1 and 3.2).

3.3.2 Spatial Distribution

The spatial distribution (also called *skew*) of errors refers to the spread of errors at various positions in a strand. Both Heckel et al [13] and `DNASimulator` assume a uniform distribution of errors. However, Fig 3.2b shows that the distribution is skewed toward the terminal ends of the strand, with a greater skew toward the end of the strand. Only the first 2 positions (0 and 1), and the last position (110) are affected; the remaining positions have approximately amount of noise. The chemical cause is likely faulty bonding during PCR, which causes primers to detach incorrectly from terminal positions of the strand.

A spatial skew for the terminal ends was used as an additional parameter for the simulation. Across all coverages, the BMA algorithm tended to converge to metrics for real data. However, the accuracy of the Iterative algorithm decreased greatly; and dropped below that of real data. This is due to the sensitivity of the Iterative algorithm to spatial distribution, as shown in §3.2 (Fig. 3.4a). Results for $N = 6$ were identical, and are provided in appendix C. The trace reconstruction accuracy results for Iterative are thus an over-correction due to the underlying error distribution introduced by the Iterative algorithm, and not by the simulator.

Note that the Hamming comparison for BMA is no longer symmetric due to the large number of errors towards the end of the strand. Both halves of the strand follow a linear trend, but the latter half has a greater baseline.



(a) Hamming errors for Iterative

(b) Gestalt-aligned errors for Iterative

(c) Hamming errors for BMA

(d) Gestalt-aligned errors for BMA

Figure 3.5: Post-Reconstruction analysis of simulated data with skew at $N = 5$

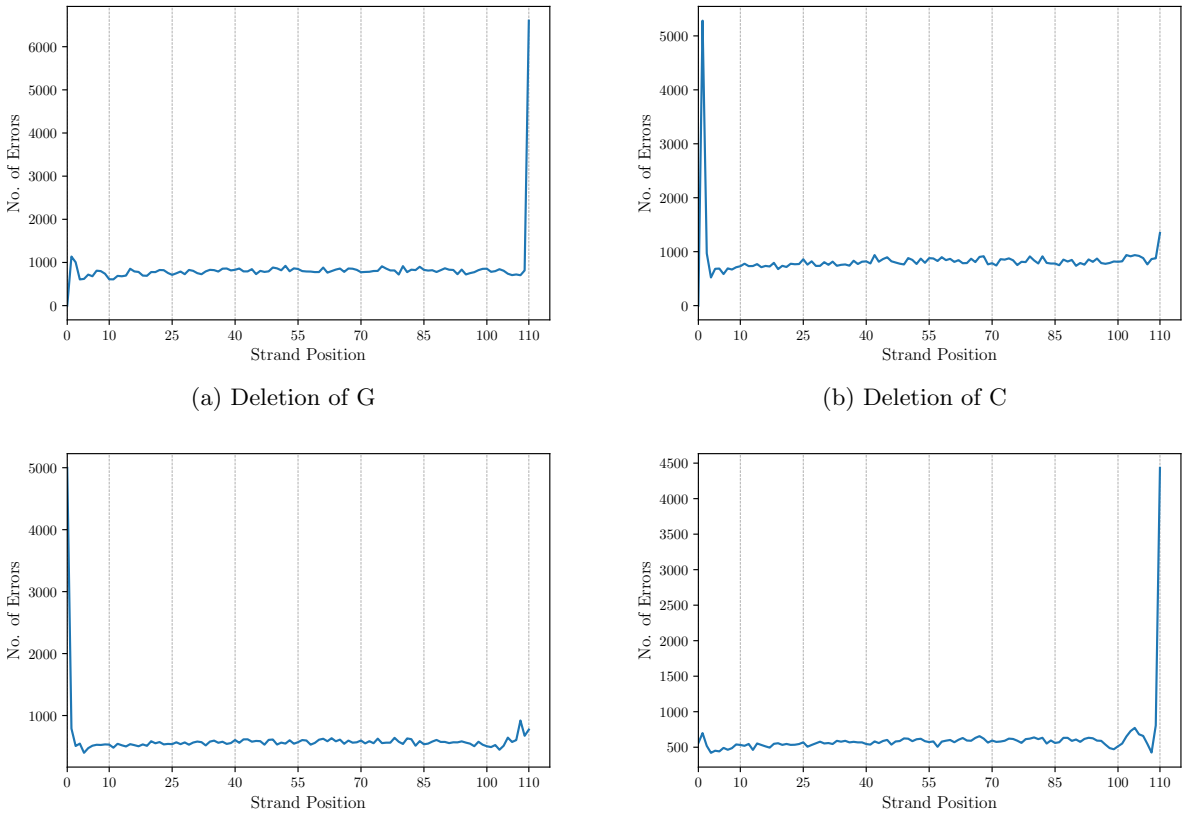
3.3.3 Second-Order Errors

Second-order errors are insertions, deletions or substitutions of specific bases, for example, the insertion of A, or the substitution of G with C. The Nanopore dataset was further analyzed with second-order errors. It was observed that the 10 most common second-order errors comprised of 56% of all errors, since all of them were single base errors involving the deletion, substitution or deletion of one base only; more complicated errors were obviously less likely.

However, it was also observed that the common second-order errors had a skew in their spatial distribution as well (Fig. 3.6), with significantly more errors at one of the terminal positions. The possible chemical reasons for this are not known, and the skew could be an artifact of the particular Nanopore dataset, instead of the general Nanopore sequencing process.

The ten most common second-order errors were included as parameters for the simulation, which resulted in a further decrease in accuracy despite the same aggregate probability (Table 3.1). The results are identical to those for §3.3.2 since there is no addition of aggregate spatial distortions, and are shown in Appendix C.

After applying this parameter to our model, both per-strand and per-character accuracies decrease for BMA, and converge to those for real data (Table 3.1). For example, at $N = 5$, the per-strand accuracy for BMA is 44.78% as compared to 29.04% for real data, and the per-character accuracy is 88.67% compared to 87.74% for real data. However, the metrics do not converge for the



(a) Deletion of G

(b) Deletion of C

(c) Insertion of C

(d) Insertion of G

Figure 3.6: Analysis of Second Order Errors in Nanopore data *before* reconstruction

Iterative algorithm, likely due to the asymmetricity of its gestalt-aligned curve (see Fig 3.5b), and its sensitivity to skewed spatial distribution of errors within a strand before reconstruction. We investigate the sensitivity of these algorithms to some parameters (spatial distribution, coverage, and aggregate error rate) below.

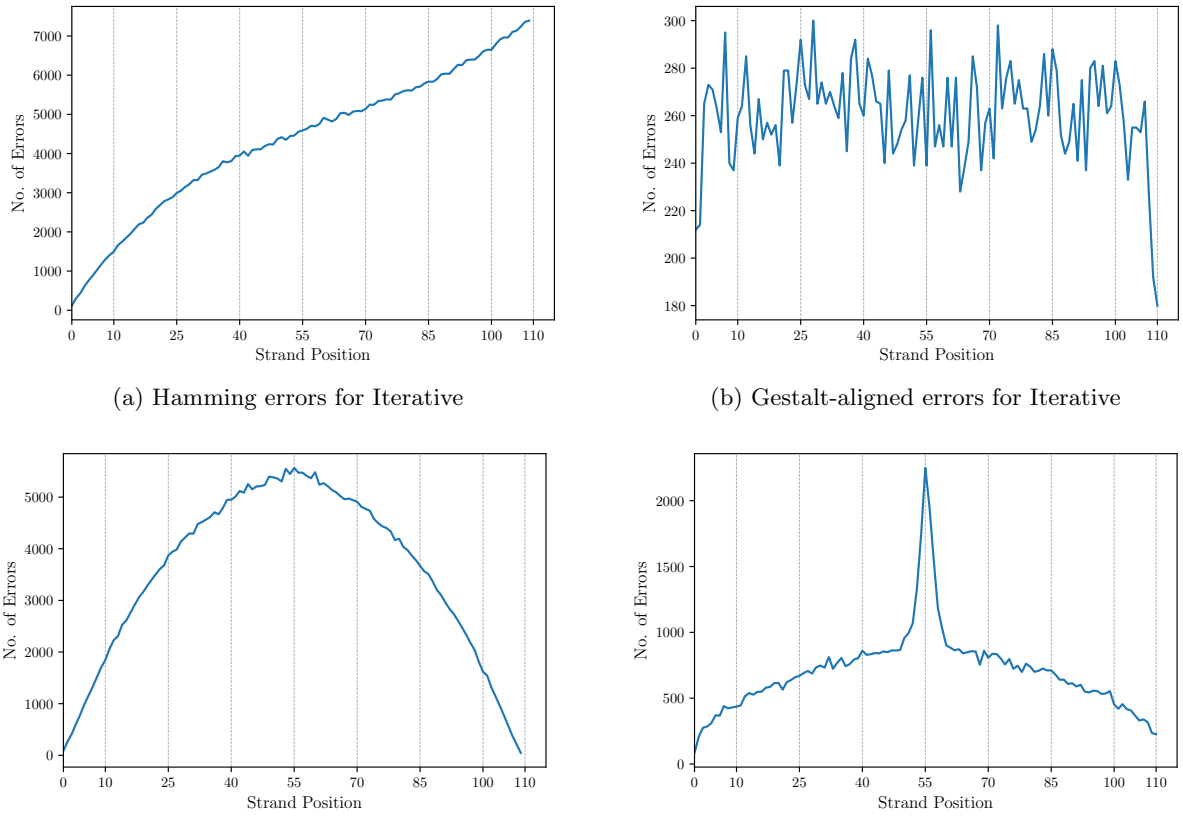
3.4 Sensitivity Analysis

In this section, we investigate the impact of different spatial distributions (uniform, A-shaped and V-shaped) and different error profiles ($\bar{p} = 0.03, 0.09, 0.12$, and 0.15) on BMA and Iterative reconstruction algorithms.

3.4.1 Varying Coverage and Error Rate at Uniform Distribution

First, we simulated data with uniform spatial distribution at various error rates (p = 0.03, 0.06, 0.09, 0.12 and 0.15) and at different coverages (n = 5, 6, and 10) for both Iterative and BMA algorithms.

The Hamming and gestalt-aligned comparisons across all coverages were similar (see Fig. 3.10 for $N = 5$). Notably, the gestalt-aligned comparison for BMA tends to get skewed to the middle at higher coverages, since the errors at terminal ends are negligible due to a higher number of noisy



(a) Hamming errors for Iterative

(b) Gestalt-aligned errors for Iterative

(c) Hamming errors for BMA

(d) Gestalt-aligned errors for BMA

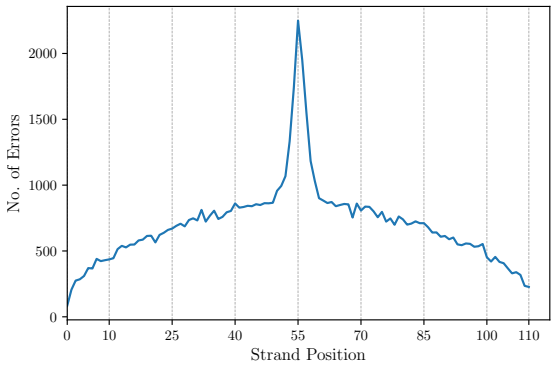
Figure 3.7: Post-Reconstruction analysis of $\bar{p} = 0.15$ data with uniform spatial distribution

copies (Fig. 3.8).

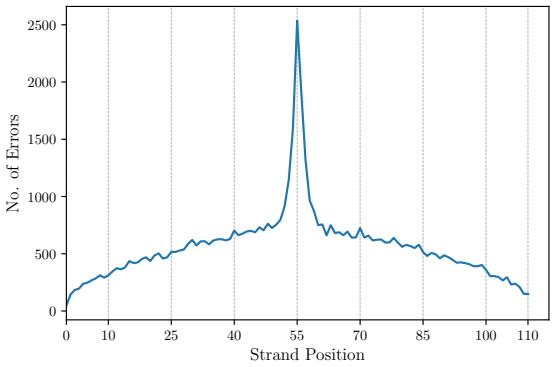
We observed that the two trace reconstruction algorithms respond differently to skews in their distribution, due to the underlying mechanism of the algorithms. For a given uniform error distribution, the BMA algorithm tends to produce a symmetric spatial distribution for a uniform error distribution, while the Iterative algorithm produces a linear spatial distribution. Further, the most common errors after Iterative reconstruction were deletion errors (90% of total).

3.4.2 Varying Spatial Distribution at Constant Error and Coverage

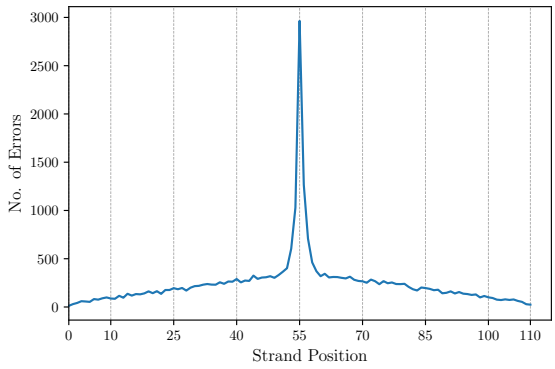
In this experiment, we simulated two datasets with a A-shaped and V-shaped spatial distribution respectively, and at aggregate $\bar{p} = 0.15$ (Fig. 3.9), and ran the BMA algorithm on the two datasets. The A-shaped curve was obtained using a triangular distribution with $a = 0, b = 0.30$ and $\bar{x} = 0.15$. The V-shaped distribution was obtained by inverting the A-shaped distribution.



(a) $N = 5$

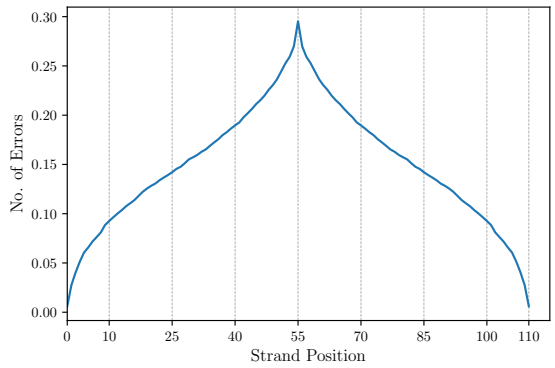


(b) $N = 6$

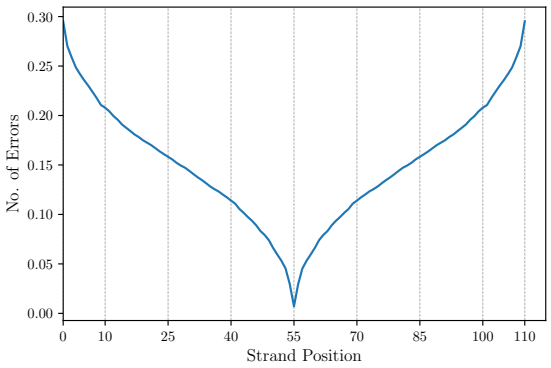


(c) $N = 10$

Figure 3.8: Post-reconstruction gestalt-aligned errors of $\bar{p} = 0.15$ data for BMA



(a) A-Shaped Curve

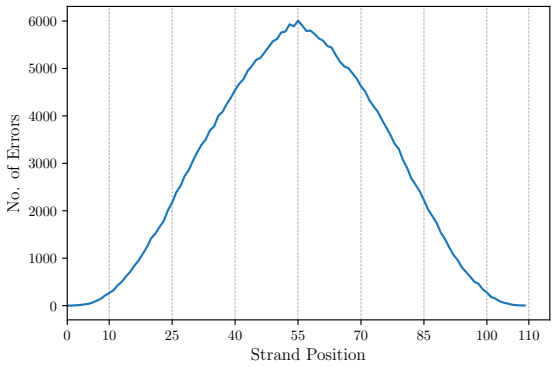


(b) V-Shaped Curve

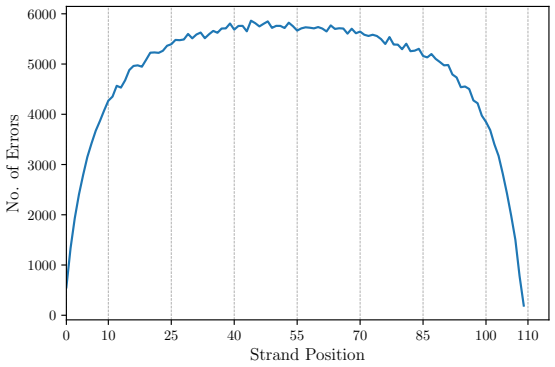
Figure 3.9: Pre-reconstruction spatial distributions at $\bar{p} = 0.15$

The BMA algorithm had a greater accuracy on strands with A-shaped distribution of errors. This is because the A-shaped curve has a higher concentration of errors in the middle of the strand, and BMA propagates errors to the middle anyway. The terminal positions are reconstructed accurately due to the lower error incidence rate and the higher fidelity of BMA at terminal positions. Consequently, the Hamming and gestalt-aligned comparisons are symmetric.

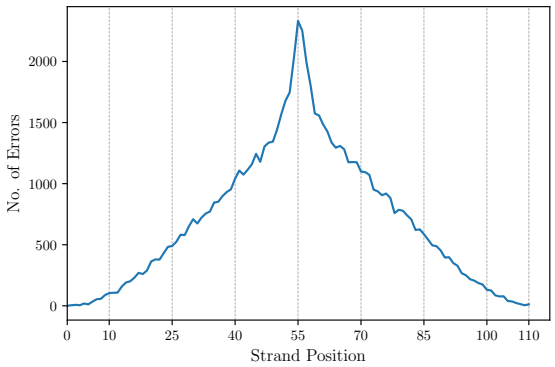
Conversely, BMA had a lower accuracy on strands with V-shaped distribution of errors. The



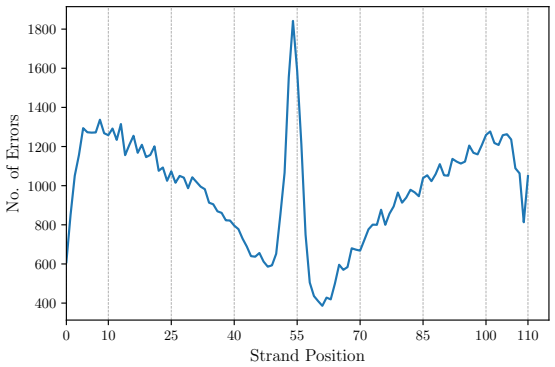
(a) Hamming errors for A-Shaped Curve



(b) Hamming errors for V-Shaped Curve



(c) Gestalt-aligned errors for A-Shaped Curve



(d) Gestalt-aligned errors for V-Shaped Curve

Figure 3.10: Post-reconstruction analysis for BMA; Data = $\bar{p} = 0.15$ data with skewed curves

Hamming and gestalt-aligned comparisons are not symmetric since there are significant errors at the terminal positions.

Chapter 4

Conclusions

4.1 Summary

In this research project, we devised and refined a simulator for noisy channels in DNA storage, and improved on `DNASimulator`, an existing simulator. We analyzed suitable metrics, datasets, and parameters for modelling the simulator. Compared to `DNASimulator`, our simulator converged closer to real data based on per-strand accuracy (15% v/s 38% difference for `DNASimulator`) and per-character accuracy (1% v/s 6% difference for `DNASimulator`) for the BMA algorithm. However, like `DNASimulator`, our simulator did not adequately converge for the Iterative algorithm.

Further, we found that the spatial distribution of errors within a strand is a key determinant of trace reconstruction accuracy; which is a factor that had not been considered by existing simulators. We used this insight to measure the impact of varying spatial distribution on the performance of trace reconstruction algorithms, and showed that the Iterative algorithm did not respond robustly to changes in spatial distribution.

4.2 Limitations and Challenges

A key limitation of our simulator is that it does not separately model the errors introduced at each stage of the DNA storage pipeline separately; it uses aggregate statistics across all stages. An ideal simulator should allow for a multi-stage, composable simulation process.

Another issue was the difficulty in choice of discriminant or metric for the simulator. While trace reconstruction accuracy is a justifiably reasonable metric, it is not clear which trace reconstruction algorithm(s) should be prioritized; in other words, whether the simulator should attempt to converge to accuracies for the BMA algorithm or the Iterative algorithm. Further, the choice of which coverage should be prioritized is also important, since the simulator might converge to real data for higher coverages while diverging at lower coverages.

4.3 Recommendations for Future Work

In order to make the simulator more robust, it should be comprehensively tested against multiple high-error datasets. This may require generalizing second-order errors to more granular parameters

such as a histogram of the counts and locations of all possible errors. However, with the addition of more parameters, it must be ensured that the simulator is able to summarize the general properties of the DNA storage pipeline, and not memorize a given dataset.

Further, a definitive metric or evaluation criteria could be devised to clearly prioritize specific algorithms at specific coverages, since it is impossible to simulate data that converges on every metric with the real data.

Finally, the insights drawn about the asymmetricity in strands reconstructed by the Iterative algorithm, its sensitivity to spatial distribution, and its linear propagation of errors to the end of the strand, could potentially be used to improve the Iterative algorithm. Possible techniques would include performing a two-way reconstruction like BMA, or using heuristics to assign a higher weightage to noisy copies that closely align with the partially reconstructed strand.

References

[1] Joseph Amankwah-Amoah. Competing technologies, competing forces: The rise and fall of the floppy disk, 1971–2010. *Technological Forecasting and Social Change*, 107:121–129, 2016.

[2] Daniella Bar-Lev, Itai Orr, Omer Sabary, Tuvi Etzion, and Eitan Yaakobi. Deep DNA Storage: Scalable and Robust DNA Storage via Coding Theory and Deep Learning, 2021.

[3] Tuundefinedkan Batu, Sampath Kannan, Sanjeev Khanna, and Andrew McGregor. Reconstructing Strings from Random Traces. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, page 910–918, USA, 2004. Society for Industrial and Applied Mathematics.

[4] James Bornholt, Randolph Lopez, Douglas M. Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A DNA-Based Archival Storage System. *SIGARCH Comput. Archit. News*, 44(2):637–649, March 2016.

[5] Luis Ceze, Jeff Nivala, and Karin Strauss. Molecular digital data storage using DNA. *Nature Reviews Genetics*, 20:456–466, 2019.

[6] Shubham Chandak, Kedar Tatwawadi, Billy Lau, Jay Mardia, Matthew Kubit, Joachim Neu, Peter Griffin, Mary Wootters, Tsachy Weissman, and Hanlee Ji. 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)*, pages 147–156, 2019.

[7] Gadi Chaykin, Nili Furman, Omer Sabary, and Eitaan Yakobi. DNA Storage Simulator. https://github.com/gadihh/DNASimulator, 2020.

[8] Jiecao Chen, He Sun, David P. Woodruff, and Qin Zhang. Communication-Optimal Distributed Clustering, 2017.

[9] George M. Church, Yuan Gao, and Sriram Kosuri. Next-Generation Digital Information Storage in DNA. *Science*, 337(6102):1628–1628, 2012.

[10] Yaniv Erlich and Dina Zielinski. DNA Fountain enables a robust and efficient storage architecture. *Science*, 355(6328):950–954, 2017.

[11] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M. LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, 2013.

[12] Robert N. Grass, Reinhard Heckel, Michela Puddu, Daniela Paunescu, and Wendelin Stark. Robust Chemical Preservation of Digital Information on DNA in Silica with Error-Correcting Codes. *Angewandte Chemie International Edition*, 54(8):2552–2555.

[13] Reinhard Heckel, Gediminas Mikutis, and Robert N. Grass. A Characterization of the DNA Data Storage Channel. *Scientific Reports*, 9(9663), 2019.

[14] Reinhard Heckel, Ilan Shomorony, Kannan Ramchandran, and David N. C. Tse. Fundamental Limits of DNA Storage Systems, 2017.

[15] Mayank Keoliya. DNA Simulator. https://github.com/mkeoliya/dna_storage_simulator, 2021.

[16] Yu Li, Sheng Wang, Chongwei Bi, Zhaowen Qiu, Mo Li, and Xin Gao. DeepSimulator1.5: a more powerful, quicker and lighter simulator for Nanopore sequencing. *Bioinformatics*, 36(8):2578–2580, 01 2020.

[17] Thomas P. Niedringhaus, Denitsa Milanova, Matthew B. Kerby, Michael P. Snyder, and Annelise E. Barron. Random access in large-scale dna data storage. *Anal Chem*, 83(12):4327–4341, 2011.

[18] Cyrus Rashtchian, Konstantin Makarychev, Miklós Rácz, Siena Dumas Ang, Djordje Jevdjic, Sergey Yekhanin, Luis Ceze, and Karin Strauss. Clustering Billions of Reads for DNA Data Storage. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3362–3373, Red Hook, NY, USA, 2017. Curran Associates Inc.

[19] John W. Ratcliff and David Metzener. Pattern matching: The Gestalt Approach, 1988.

[20] Omer Sabary, Yoav Orlev, Roy Shafir, Leon Anavy, Eitan Yaakobi, and Zohar Yakhini. SOLQC: Synthetic Oligo Library Quality Control tool. *Bioinformatics*, 37(5):720–722, 08 2020.

[21] Omer Sabary, Alexander Yucovich, Guy Shapira, and Eitan Yaakobi. Reconstruction Algorithms for DNA-Storage Systems. *bioRxiv*, 2020.

[22] Michael Schwarz, Marius Welzel, Tolganay Kabdullayeva, Anke Becker, Bernd Freisleben, and Dominik Heider. MESA: automated assessment of synthetic DNA fragments and simulation of DNA synthesis, storage, sequencing and PCR errors. *Bioinformatics*, 36(11):3322–3326, 03 2020.

[23] Chen Yang, Justin Chu, René L Warren, and Inanç Birol. NanoSim: nanopore sequence read simulator based on statistical characterization. *GigaScience*, 6(4), 02 2017. gix010.

[24] S. Tabatabaei Yazdi, Ryan Gabrys, and Olgica Milenkovic. Portable and Error-Free DNA-Based Data Storage. *Scientific Reports*, 7:5011–5031, 2017.

[25] S. Tabatabaei Yazdi, Y. Yuan, and J. Ma. A Rewritable, Random-Access DNA-Based Storage System. *Scientific Reports*, 5(1):2045–2322, 2015.

[26] Zhenhua Yu, Fang Du, Rongjun Ban, and Yuanwei Zhang. SimuSCoP: reliably simulate Illumina sequencing data based on position and context dependent profiles . *BMC Bioinformatics*, 21(331):3115–3125, 07 2020.

[27] Jinny X. Zhang, Boyan Yordanov, Alexander Gaunt, and Michael X. Wang. A deep learning model for predicting next-generation sequencing depth from DNA sequence. *Nature Communications*, 12(4387), 07 3032.

[28] Victor Zhirnov, Reza M. Zadegan, Gurtej S. Sandhu, George M. Church, and William L. Hughes. Nucleic acid memory. *Nature Materials*, 15:366–370, 2016.

Appendix A

Code Repository

The code for our project can be found on GitHub [15], and can be accessed via this link. It is supplemented with a guide on how to set up the codebase, process data, conduct system testing, and replicate the findings of this project. The repository contains the naive simulator, a fork of `DNASimulator`, the Microsoft Nanopore datset, as well as an implementation of the trace reconstruction algorithms used in this project. Jupyter notebooks are included to enable quick prototyping and visualization of the data.

Note that all experiments were performed on the `rachmaninoff2.d2.comp.nus.edu.sg` server at the School of Computing, NUS. The `clang` C compiler is recommended for compiling C files, and `Python 3.7` for running the Python scripts under `Scripts/`.

Appendix B

Edit Distance Operation Algorithm

Given a reference string s_1 and a noisy copy s_2, the following algorithm computes the minimum edit distance operations required to transform s_1 into s_2. The types of operations are `INS`, `DEL`, and `SUBS`. The source code is available here.

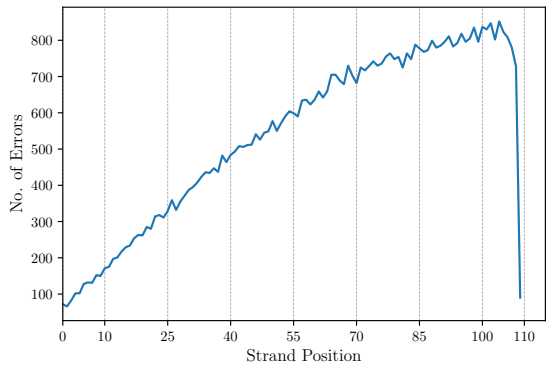
Algorithm 2 Computing edit distance operations

$m \leftarrow 0$ ▷ m denotes the current position in s_1
$n \leftarrow 0$ ▷ n denotes the current position in s_2
Function `EditDistanceOps`(s_1, s_2, m, n)**:**
  if $m == 0$ **then**
    $ops \leftarrow [\,]$
    for $k \leftarrow$ *1 to* N **do**
      $ops.insert($`DEL`$)$
    end
    return ops
  end
  if $n == 0$ **then**
    $ops \leftarrow [\,]$
    for $k \leftarrow$ *1 to* M **do**
      $ops.insert($`INS`$)$
    end
    return ops
  end
  if $s_{1m-1} == s_{2n-1}$ **then**
    $ops \leftarrow$ `EditDistanceOps`$(s_1[:m-1], s_2[:n-1], m-1, n-1)$
    $ops.insert($`EQUAL`$)$
    return ops
  end
  else
    $ops_1 \leftarrow$ `EditDistanceOps`$(s1[:m-1], s2[:n], m-1)$
    $ops_2 \leftarrow$ `EditDistanceOps`$(s1[:m], s2[:n-1], m, n-1)$
    $ops_3 \leftarrow$ `EditDistanceOps`$(s1[:m-1], s2[:n-1], m-1, n-1)$
    $ops \leftarrow$ `ChooseRandomAndInsertOp`(ops_1, ops_2, ops_3)
    return ops
  end

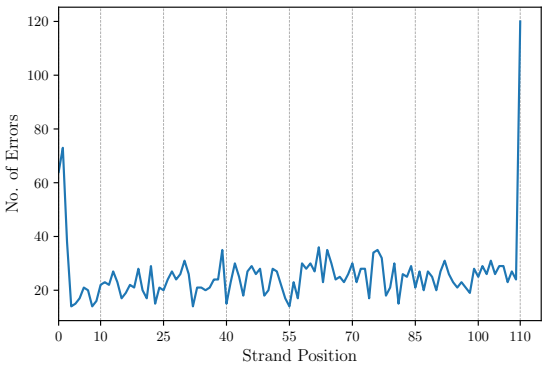
Appendix C

Additional Figures

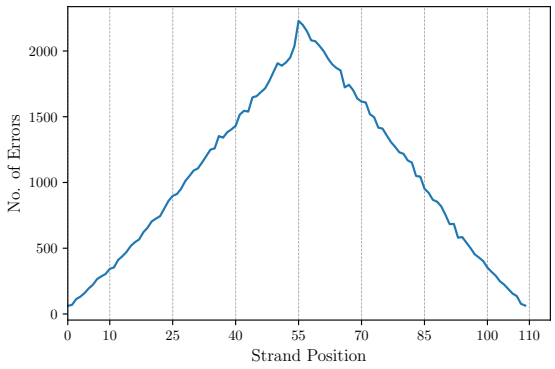
C.1 Analysis of Nanopore Data after Reconstruction



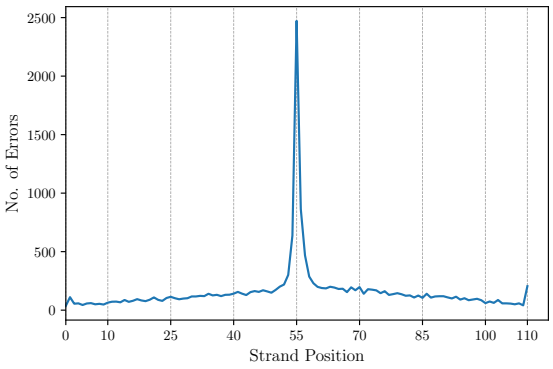
(a) Hamming errors for Iterative



(b) Gestalt-aligned errors for Iterative



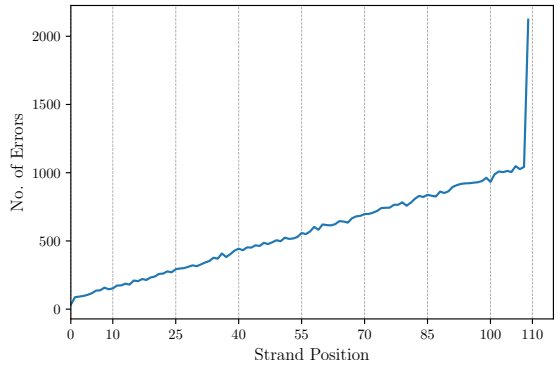
(c) Hamming errors for BMA



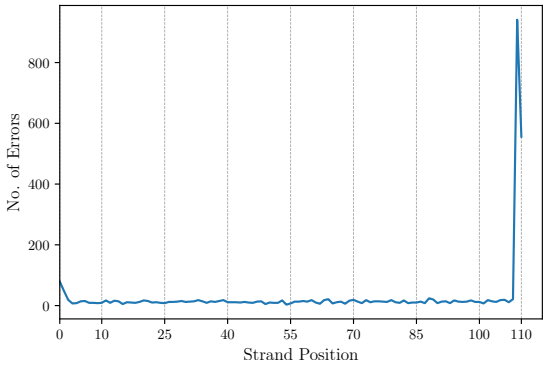
(d) Gestalt-aligned errors for BMA

Figure C.1: Post-Reconstruction analysis of Nanopore data at $N = 6$

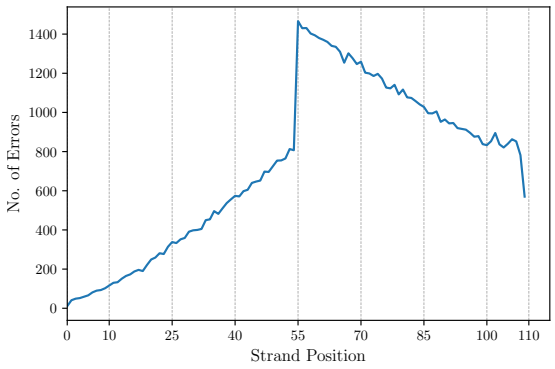
C.2 Analysis of Simulated Data with Skew



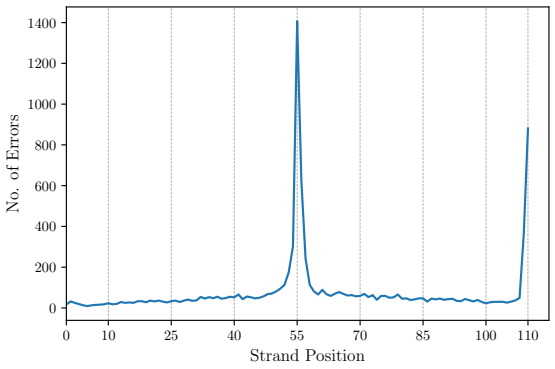
(a) Hamming errors for Iterative



(b) Gestalt-aligned errors for Iterative



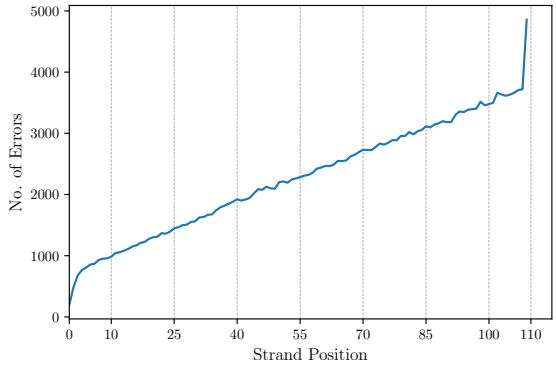
(c) Hamming errors for BMA



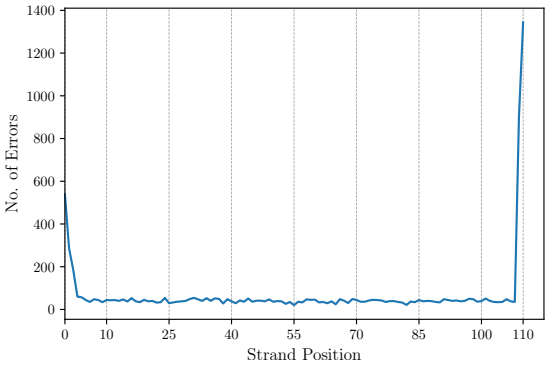
(d) Gestalt-aligned errors for BMA

Figure C.2: Post-Reconstruction Analysis of Simulated Data with Skew at $N = 6$

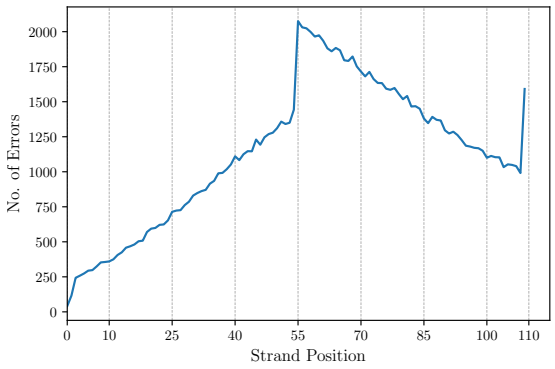
C.3 Analysis of Simulated Data with Second-Order Skew



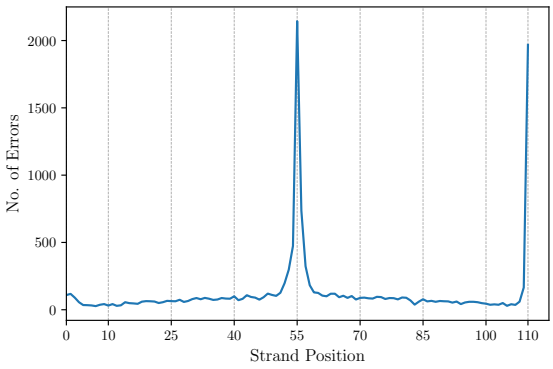
(a) Hamming errors for Iterative



(b) Gestalt-aligned errors for Iterative



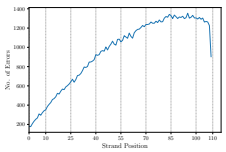
(c) Hamming errors for BMA



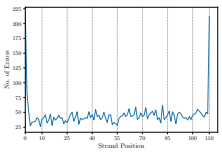
(d) Gestalt-aligned errors for BMA

Figure C.3: Post-reconstruction analysis of simulated data with second-order errors at $N = 5$

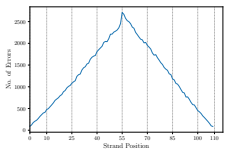
C.4 Overall Post-Reconstruction



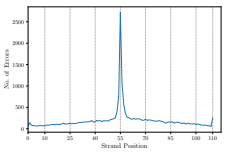
(a) Hamming errors for Iterative



(b) Gestalt-aligned errors for Iterative

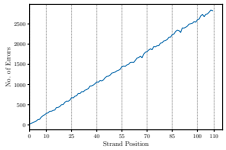


(c) Hamming errors for BMA

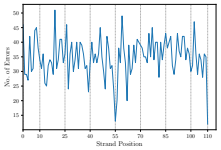


(d) Gestalt-aligned errors for BMA

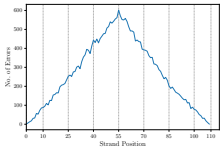
Figure C.4: Real Nanopore at $N = 5$



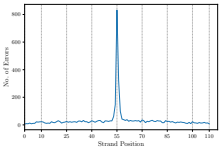
(a) Hamming errors for Iterative



(b) Gestalt-aligned errors for Iterative

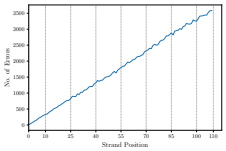


(c) Hamming errors for BMA

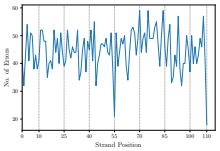


(d) Gestalt-aligned errors for BMA

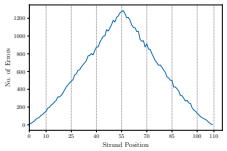
Figure C.5: Naive at $N = 5$



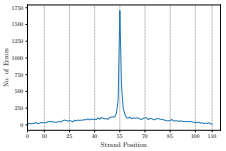
(a) Hamming errors for Iterative



(b) Gestalt-aligned errors for Iterative

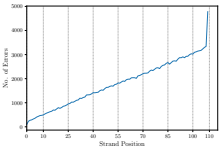


(c) Hamming errors for BMA

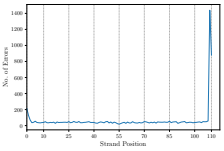


(d) Gestalt-aligned errors for BMA

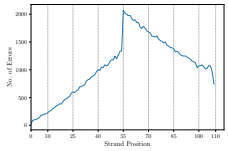
Figure C.6: Naive + Cond + LD at $N = 5$



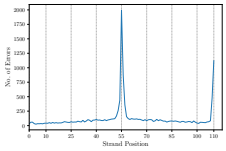
(a) Hamming errors for Iterative



(b) Gestalt-aligned errors for Iterative

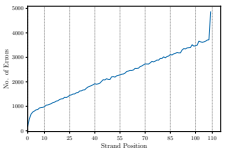


(c) Hamming errors for BMA

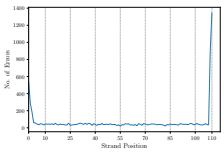


(d) Gestalt-aligned errors for BMA

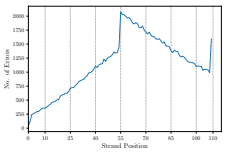
Figure C.7: Naive + Cond + LD + Skew at $N = 5$



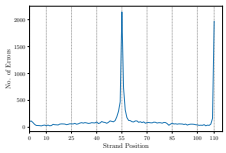
(a) Hamming errors for Iterative



(b) Gestalt-aligned errors for Iterative



(c) Hamming errors for BMA



(d) Gestalt-aligned errors for BMA

Figure C.8: Naive + Cond + LD + Skew + Second-order errors at $N = 5$