UNIVERSITY OF CALIFORNIA,
IRVINE

# Coding Theory in Storage and Network

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Engineering

by

Peng Fei

2021

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ALGORITHMS

# ACKNOWLEDGMENTS

It is my great pleasure to express my gratitude to my mentor, advisor and guide Professor Zhiying Wang, whose expertise was inestimable in my research path. Her dedication and intense interest on research had been mainly responsible for completing my work. I would always remember her wisdom and engagement to greatness in my future career.

I would like to thank Professor Zhou Li to co-advise me in the security data compression project. This interdisciplinary experience is invaluable for my career path.

I would like to thank Professor Professor Fadi Kurdahi for being my faculty contact and my first year advisor . His great expertise has lead me to the research path and supported me find my own research passion.

I would like to thank Professor Hamid Jafarkhani for serving on my dissertation committee, and Professor Sandmeyer, Suzanne for serving on my qualifying examination committee. I appreciate all these valuable suggestions from my committee members. I am thankful to all the professors I engaged with, either through courses or personally, over the past five years at UCI.

I would like to thank Dr. Lily Wu for being my friend and life mentor. Her kindness and friendship gave me so much support to accommodate the life in the United State as an international student. I also thank her for hiring me as her teaching assistant. I would not reach this far without her help.

It has been a great pleasure to have co-authored with Professor Syed A. Jafar, Professor Kangkook Jee, Professor Ding Li, Xiao Yu, Zhen chen and Keqing Fu. I really enjoy the time working with them. My thanks extend to former and current colleagues including Professor Lawrence Kulinsky, Davit Hovhannisyan, Marwen Zorgui, Alireza Javani, Weiqi Li, Wenkai Zhang, Xiaoran Li and Keqing Fu. I am proud of being a part of such a wonderful group.

I would like to thank my classmates and friends, Chenhui Li, YuanHung Lin, Yucheng Tang, Jad georges khalifeh, Linyi Xia, Yicheng Zhang, Jinrun Hu, Jialie Yan, Junyi Li, Rongyu Cao. Thanks for their accompanies and friendships, which makes my life more colorful.

Finally, I must express my gratitude to my parents. I would not reach this far without their guidance and endless support. I would especially like to thank my love and friend, Xinyi Nie, who has stood by me through all my travails, my absences, my fits of annoyance and impatience. She gave me support and help, discussed ideas and prevented several wrong turns.

# VITA

## Peng Fei

### EDUCATION

**Doctor of Philosophy in Computer Engineering** **2016**
University of California, Irvine *Irvine, California*

**Bachelor of Science in Software Engineering** **2012**
Dalian University of Technology *Dalian, China*

### RESEARCH EXPERIENCE

**Graduate Research Assistant** **2016–2021**
University of California, Irvine *Irvine, California*

### TEACHING EXPERIENCE

**Teaching Assistant** **2017–2020**
University of California, Irvine *Irvine, California*

# ABSTRACT OF THE DISSERTATION

Coding Theory in Storage and Network

By

Peng Fei

Doctor of Philosophy in Computer Engineering

University of California, Irvine, 2021

Assistant Professor Zhiying Wang, Chair

The high demand of data storage and data communication brings many new challenges and concerns, including but not limited to, how to efficiently store data in devices and how to reliably communicate data between different devices to avoid possible errors. Those two problems correspond to the two most important concepts in information theory: source coding and channel coding, where the former compresses data to reduce its size, and the latter enlarges the data to combat errors. In this thesis, four problems about source coding and channel coding are presented with applications in information storage and networks.

For source coding, `SEAL` is presented, which is a novel data compression approach for system log data and supports causality analysis for system security. Based on information-theoretic observations on system event data, the approach achieves lossless compression and supports near real-time retrieval of historic events. In the compression step, the causality graph induced by the system logs is investigated, and abundant edge reduction potentials are explored. In the query step, for maximal speed, decompression is opportunistically executed. `SEAL` greatly alleviates the exponentially-growing storage demand for causality analysis in industrial computer security.

For channel coding, an error-correcting code based on low-density parity-check code(LDPC) for DNA storage is first introduced. In this work, DNA storage that uses affordable and portable nanopore sequencing is considered as the reading mechanics. Unlike traditional data storage systems, errors occur asymmetrically among the four types of nucleotide bases of DNA. Quaternary codes can be employed for error correction, but suffer from high complexity. For this problem, a turbo-like decoder for the DNA storage channel is designed. Meanwhile, the corresponding density evolution algorithms are designed to prove the optimal bound for the decoders. Simulation results show that the binary LDPC codes have a similar bit error rate but with a speedup by a factor of 4 compared to quaternary codes.

Next, a source of synchronization errors in DNA storage is identified, which could result in missing symbols in the read result from nanopore sequencing. To limit such errors, constrained codes are presented. Moreover, this work shows encoding algorithms and maximum a posteriori decoding algorithms in the presence of additive Gaussian noise and deletions. The simulation shows a trade-off between the coding rate and the missing-symbol rate. The decoding simulation results show that the algorithms can correct missing-symbols error efficiently. Meanwhile, simulated data from DeepSimulator by Y. Li et al. [67] and the real-world data also show that the constrained DNA strings have fewer missing-symbol errors.

Finally, coding for network synchronization is investigated. This work studies the problem of clock synchronization in an arbitrary network, viewed as a graph. Each server is a node, and two nodes are connected by an edge if their time discrepancy (edge weight) is measured. The time discrepancy is known by one or both of these two servers. The goal is to reduce the communication cost between server nodes and the master node, which collects the discrepancy information to eliminate the loop-wise offset surplus in the network. For the two important cases where each time discrepancy is known by both adjacent servers and by only one of the adjacent servers, the optimal schemes are found.

# Chapter 1

# Introduction

With the coming of modern information age, the sharp increase in the amount of information raises high demands for data storage and management. While technology enables the increase of the storage capacity of the current devices such as hard disk, solid-state drives, and random access memory, other possible storage devices and materials may offer orders of magnitude improvement in storage density, for example, DNA storage. Moreover, is it common to coordinate multiple storage devices together to build the data center and coordinate multiple data centers together to build the data center network. During those procedures, many new challenges and concerns have appeared, including but not limited to, how to efficiently store data in devices, and how to reliably communicate data between different devices to avoid possible errors. Those two problems correspond to the two most important concepts in the information theory: source coding and channel coding. In 1948, Shannon stated the source coding theorem and channel coding theorem [101] that quantify the fundamental limits of data compression and data communication, respectively.

This thesis introduces four works about coding theory with application in data storage and networks. Chapter 2 introduces a work about the usage of source coding in the security analysis field, Chapter 3 and 4 introduce two works about the usage of channel coding in DNA storage and chapter 5 introduces a work about the usage of network coding for network clock synchronization.

## 1.1 Source coding for security analysis

Causality analysis automates attack forensic and facilitates behavioral detection by associating causally related but temporally distant system events. Despite its proven usefulness, the analysis suffers from the innate big data challenge to store and process a colossal amount of system events that are constantly collected from hundreds of thousands of end-hosts in a realistic network. In addition, the effectiveness of the analysis to discover security breaches relies on the assumption that comprehensive historical events over a long span are stored. Hence, it is imminent to address the scalability issue in order to make causality analysis practical and applicable to the enterprise-level environment.

In Chapter 2, we present `SEAL`, a novel data compression approach for causality analysis. Based on information-theoretic observations on system event data, our approach achieves lossless compression and supports near real-time retrieval of historic events. In the compression step, the causality graph induced by the system logs is investigated, and abundant edge reduction potentials are explored. In the query step, for maximal speed, decompression is opportunistically executed. Experiments on two real-world datasets show that `SEAL` offers 2.63x and 12.94x data size reduction, respectively. Besides, 89% of the queries are faster on the compressed dataset than the uncompressed one, and `SEAL` returns exactly the same query results as the uncompressed data.

## 1.2 LDPC code in DNA storage

DNA becomes an attractive storage medium in recent years for its ultra-high density, millennia-long endurance, and efficient replication. In Chapter 3 we consider DNA storage that uses the affordable and portable nanopore sequencing as the reading mechanics. Unlike traditional data storage systems, errors occur asymmetrically among the four types of nucleotide bases of DNA. Quaternary codes can be employed for error correction, but suffer from high complexity. In this work, we design binary LDPC codes with a turbo-like decoder for the DNA storage channel. We also design the corresponding density evolution algorithms to prove the optimal bound for our decoders. Simulation results show that our binary LDPC codes have a similar bit-error rate but with a speed up by a factor of 4 compared to quaternary codes.

## 1.3 Constrained code in DNA storage

As mentioned, Nanopore sequencing is an attractive reading technology for DNA storage due to the ease of access. However, sequencing errors greatly affects the reliability of the stored information. In Chapter 4, we identify a source of synchronization errors, which could result in missing symbols in the read-out sequences. Then we present constrained codes to limit such errors. We show encoding algorithms and maximum a posteriori decoding algorithms in the presence of additive Gaussian noise and deletions.

Our simulation shows a tradeoff between the coding rate and the error rate. The experiment over DeepSimulator [67] data and the real world data also shows that the encoded DNA strings have less errors during the nanopore sequencing. Specifically, our encoded data improve deletion rate of base-calling accuracy of DeepSimulator from 1.76% to 0.59%. The decoding simulation results show that the algorithms can correct missing-symbols error efficiently.

## 1.4 Channel Coding for Clock Synchronization

In Chapter 5, we study the problem of clock synchronization in arbitrary networks, viewed as a graph. Every pair of adjacent servers has a time discrepancy (edge weight) that is agreed by both servers and only known by these two adjacent servers. Server nodes send this time discrepancy information to a master node. After collecting the information, the master node aims to coordinate otherwise independent clocks of servers, i.e., to eliminate the loop-wise offset surplus in the network. We focus on the communication cost between server nodes and the master node. For the two important cases where each time discrepancy is known by both adjacent servers and by only one of the adjacent servers, we find the optimal schemes. It is interesting to note that for the former case, our scheme is also straggler (slow or fail server) robust. For the rest of the cases, we propose an algorithm that outperforms the trivial solution.

# Chapter 2

# SEAL: Storage-efficient Causality Analysis on Enterprise Logs with Query-friendly Compression

## 2.1 Introduction

System logs constitute a critical foundation for enterprise security. The latest computer systems have become more and more complex and interconnected, and attacker techniques have advanced to take advantage and nullify the conventional security solutions which are based on static artifacts. As a result, the security defense has turned more to pervasive system event collection in building effective security measures. Research has extensively explored security solutions using system logs. *Causality analysis* in the log setting (or *attack provenance*), as defined in [122], is one such direction that reconstructs information flow by associating interdependent system events and operations. For any suspicious events, the analysis automatically traces back to the initial penetration (root-cause diagnosis), or measures the amount of the impact by enumerating the system resources affected by the attacker (attack ramification). Encouragingly, the security solutions based on pervasive system monitoring and causality analysis no longer remain as a research prototype. Many proposed ideas have actualized as commercial solutions[24, 34, 7].

However, due to their data-dependent nature, the effectiveness of the above security solutions is heavily constrained by the system's data storage and processing capability. On one hand, keeping large volumes of comprehensive historical system events is essential, as the security breach targeting an enterprise tends to stay at the network over a long span: an industry report by TrustWave [116] shows, on average, an intrusion *prolongs over 188 days* before the detection. On the other hand, the size of a typical enterprise network and the amount of system logs each host generates could put high pressure on the security solutions. For instance, our industrial partner reported that on average *50 GB amount of logs are produced from a group of 100 hosts daily*, and they can only sustain *at most three months of data* despite the inexpensive storage cost. There is a compelling need for a solution that can scale storage and processing capacity to meet the enterprise-level requirement.

**Lossless compression versus lossy reduction.** Compression techniques [117] come in handy for improving the storage efficiency of causality analysis. Existing approaches [111, 122, 62, 52] tend to carry out *lossy reduction*, which *removes* logs matching pre-defined patterns, leading to unavoidable information loss. Although they showed that the validity of causality analysis is preserved on samples of investigation tasks, *there is no guarantee that every task will derive the right outcome.* In Section 2.2.3, we show examples about when they would introduce false positives/negatives. In addition, the accuracy of other applications such as behavioral detection [74, 48] and machine-learning based anomaly detection [88, 89, 125, 12, 29, 69] would be tampered, when they use the same log data. Alternatively, *lossless compression* [117] allows *any* information to be restored and thus causality analysis is preserved. Though the standard tools like Gzip [28] are expected to achieve a high compression rate, they are not applicable to our problem, because high computation overhead of *decompression* will be incurred when running causality analysis.

In this work, we challenge the common belief that lossless compression is inefficient for causality analysis, by developing `SEAL` (**S**torage-**E**fficient **A**nalysis on enterprise **L**ogs) under information-theoretic principles. Compared to the previous approaches, logs under a wider range of patterns can be compressed in a lossless fashion without the need for carefully examining conditions such as traceability equivalence or dependence preservation, while the validity and efficiency of any investigation task of causality analysis are preserved.

**Contributions.** The main contributions of this paper are as follows.

• We develop a framework of *query-friendly compression* (`QFC`) specialized for causality analysis. In this framework, the dependency graph is induced from the logs, and lossless compression is applied to the structure (vertices and edges) and then to the edge properties, or attributes (e.g., timestamp). `QFC` ensures every query is answered accurately, while the query efficiency is guaranteed as the majority of operations required by queries are done *directly on the compressed data.*

• We design compression and querying algorithms according to the definition of `QFC`. For graph structures, we define merge patterns to be subgraphs whose edges are combined into one new edge. For edge properties, delta coding [81] and Golomb codes[45] are applied to exploit temporal locality, meaning that consecutively collected logs have similar timestamps. To return answers to a causality query, the proposed method obviates decompression unless the relationship between the timestamps of a compressed edge and the time range of the query cannot be determined.

• A compression ratio estimation algorithm is provided to facilitate the decision of using the compressed or uncompressed format for a given dataset. We show that the compression ratio can be determined by the average degree of the dependency graph. Our algorithm estimates the average degree by performing random walk on the dependency graph with added self-loops, and randomly restarting another walk during the process. If the estimated compression ratio of a given dataset is smaller than a specified threshold, compression can be skipped.

• The above algorithms are implemented in `SEAL`, which consists of the compression system that is applied to online system logs and the querying system that serves causality analytics. Due to the large amount of merge patterns in the dependency graphs, `SEAL` can compress online log data into a significantly smaller volume. In addition, the query-friendly design reduces the required decompression operations. We evaluate `SEAL` on system logs from 95 hosts provided by our industrial partner. The experiment results demonstrate an average of 9.81x event reduction, 2.63x storage size reduction. Besides, 89% of the queries are faster on the compressed dataset than the uncompressed one. We also evaluate `SEAL` on DARPA TC dataset [25] and achieved 12.94x size reduction. Causality analysis to investigate attacked entities is shown to return accurate results with our compression method.

## 2.2 Background

We first describe the concepts of system logs and causality analysis. Then, we review the existing works based on lossy reduction and compare `SEAL` with them.

### 2.2.1 System Logs

To transparently monitor the end-host activities in a confined network, end-point detection and response (EDR) has become a mainstream security solution [51]. A typical EDR system deploys data collection sensors to collect the major system activities such as *file*, *process* and *network* related events, as well as events with high security relevancy (e.g., login attempts, privilege escalation). Sensors then stream the collected system events to a centralized *data back-end*. Data collection at end-host hinges on different operating systems' (OS) kernel-level supports for system call level monitoring [79, 95, 13].

In this study, we obtained a dataset from the real-world corporate environment. Data sources are the system logs generated by kernel audit [95] of Linux hosts and Event Monitoring for Windows (ETW) [79] of Windows hosts respectively. The system events belong to three different categories: (i) process accesses (reads or write) files (P2F), (ii) process connects to or accepts network sockets (P2N), and (iii) process creates other processes, or exits it executions. These system events captured from each end-host are transferred to the back-end and represented in a graph data structure [60] where nodes represent system resources (i.e., process, file, and network socket) and edges represent interactions among nodes. Our system labels edges with attributes specific to system operations. For instance, amounts of data transferred for file and network operations, command-line arguments for process creations. The dataset comprises of various workloads that range from simple administrative tasks to heavy-weight development and data analysis tasks and also includes end-user desktops and laptops as well as infra-structural servers.

Among the three categories of system events (file, network, and process) in the dataset, file operations account for the majority, taking over 90% portions, therefore become the primary target for `SEAL` compression. In particular, the *file operation* like create, open, read, write or delete is logged in each file event, alongside its *owner process*, *host ID*, *file path*, and *timestamp*. All file events have been properly anonymized (no user identifiable information exists in any field of the table) to address privacy concerns.

Despite its improved visibility, data collection for in-host system activity results in a prohibitive amount of processing and storage pressures, compared to other network-level monitoring appliances [89]. For instance, our data collection deployment on average reported approximately 50 GB amount of logs for a group of 100 hosts daily. Given that a typical enterprise easily exceeds hundreds of thousands of hosts for its network, it is imminent to address the scalability issues in order to make causality analysis practical and applicable to a realistic network.

### 2.2.2 Causality Analysis in the Log Setting

After the end-point logs are gathered and reported to the data processing back-end, different applications are run atop to produce insights to security operators, such as machine-learning based threat detection [12], database queries [37, 39, 38] and causality analysis (or data provenance) [122]. Although our approach mainly focuses on the causality analysis, which requires high fidelity on its input data, it also benefits other analyses as our approach reduces data storage and computational costs.

To its core, causality analysis automates the data analysis and forensic tasks by correlating data dependency among system events. Using the restored causality, security operators accelerate *root cause* analysis of security incident and attack ramification. The causality analysis is considered to be a *de facto* standard tool for investigating long-running, multi-stage attacks, such as Advanced Persistent Threat (APT) campaigns [80]. For any suspicious events reported by users or third-part detection tools, the operator can issue a query to investigate causally related activities. The causality analysis then consults to its data back-end to restore the dependencies within the specified scope. The accuracy of causality analysis relies on the completeness of data collection, and the analysis response time and usability depend on the data access time. In Section 2.3.1, we demonstrate a causality analysis where our compression approach addresses the scalability issues without deteriorating accuracy and usability.

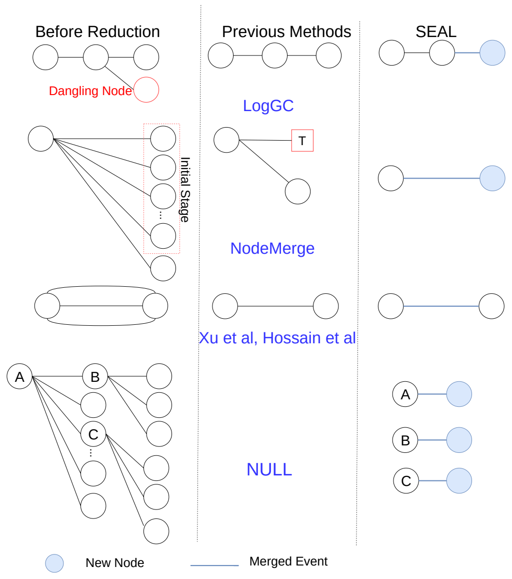


Figure 2.1: Comparison of our method `SEAL` to LogGC [62], NodeMerge [111], methods by Xu et al. [122] and Hossain et al. [52]. In NodeMerge (the second graph in the middle column), the node T represents a new node. In `SEAL` (the right column), the blue solid circles represent new nodes.

### 2.2.3 Comparison with Lossy Reduction

To reduce the storage overhead in supporting causality analysis, prior works advocated *lossy reduction* [62, 111, 122, 52], which removes logs of certain patterns before they are stored by the back-end server. Here we show the reduction rules of the prior works and compare their scope to `SEAL`.

LogGC [62] removes temporary files from the collected data that are deemed not affecting causality analysis. NodeMerge [111] merges the read-only events (`Read` events in our data) during the process initialization. The approach proposed by Xu et al. [122] removes repeated edges between two objects on the same host (e.g., multiple read events between a file and a process) when a condition termed trackability equivalence is satisfied. Hossain et al. [52] relaxes the condition of [122] such that more repeated events (e.g., repeated events *cross hosts*) can be pruned, which tends to be more conservative to maintain graph trackability.

`SEAL` is more general compared to any of the existing works. Our *lossless compression* schema

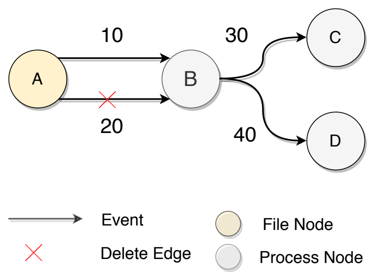


Figure 2.2: A dependency graph (see Section 2.3.1) under Full Dependency (FD) preservation reduction [52], where one edge is removed. The timestamp of each event is labeled on the edge. When querying for nodes dependent on Node $A$ after time 15, the reduced dataset returns the empty set but the original dataset returns $B, C, D$. From the example we see that FD can return less number of nodes for causality analysis under time constraints.

is agnostic to file types and is therefore complementary to LogGC. `SEAL` also processes `Write` and `Execute` events, compared to NodeMerge, and therefore covers the whole life-cycle of a process. Compared to Xu et al. and Hossain et al., `SEAL` is more aggressive, e.g., merging not only the edges repeated between a pair of nodes. Figure 2.1 also illustrates the differences. In Section 2.5, we compare the overall reduction rate, with Hossain et al., which is the most recent work.

In terms of data fidelity, none of the prior works can guarantee false negative/positive would not occur during attack investigation. For LogGC, if the removed temporary files are related to network sockets, data exfiltration done by the attacker might be missed. For NodeMerge, the authors described a potential evasion method: the attacker can keep the malware waiting for a long time before the actual attack, so that the malware might be considered as a read-only file as determined by their threshold and break the causality dependencies (see [111] Section 10.4). PCAR of Xu et al. introduces false connectivity in two (out of ten) investigation tasks (see [122] Section 4.3). Similarly, false negatives could be introduced to the system of Hossain et al. when the query has a time constraint, and we provide an example in Figure 2.2.

On the other hand, as `SEAL` ensures the completeness of logs, it can mitigate any of the above issues.

| Field | Exemplar Value |
|---|---|
| starttime | 1562734588971 |
| endtime | 1562734588985 |
| srcid | 15 |
| dstid | 27 |
| agentid | -582777938 |
| accessright | Execute |

Table 2.1: On example entry of `FileEvent`.

## 2.3 Log Compression

`SEAL` aims to compress the *dependency graph* constructed from system logs, as illustrated in Figure 2.3, while supporting the causality analysis without sacrificing query efficiency and analysis accuracy. If every analysis task results in decompressing a large portion of data, the goal of query efficiency will not be achieved. If compression causes significant information loss, the forensic analysis might lead to incorrect conclusion. Therefore, we design `SEAL` to compress the vertices and edges of a large amount of redundant information, and the compressed sets of edges are chosen such that we can restrain the frequency or overhead of decompression.

In this section, we first describe the dataset to be processed and the query to be run by an analyst. Then, we introduce the concept Query-friendly Compression (`QFC`) and show how it can be applied to system logs. Moreover, we introduce the compression algorithms that can be applied on the logs and compare them with the prior research. Finally, we propose an algorithm to estimate the compression ratio based on which one can determine when to compress.

### 2.3.1 Dataset and Event Query

Table 2.1 shows the primary dataset (`FileEvent`) we need to compress and the main fields. The start and end timestamp of each event are logged by `starttime` and `endtime`. An event links a source object and a destination object, distinguished by `srcid` (Source ID) and `dstid` (Destination ID). The object associated with each event can be `file` or `process`. All events occur within a host, denoted by `agentid`, and there is no cross-host event. There are three types of operations associated with an event, including `Execute`, `Read` and `Write`, recorded by `accessright`. To notice, the properties of objects, like the filenames and paths, are stored in other tables. But because the other tables' volume is small, we do not process them specifically.

**Causality analysis on `FileEvent`.** We assume that a dependency graph $G = (V, E)$ can be derived from `FileEvent`, in which the vertices ($V$) are the objects and the *directed* edges ($E$) are the events. Causality analysis uncovers the causality dependency of edges, and we define its computation paradigm below, in a way similar to the definition from Wu et al. [122].

***Definition 1 (Causality dependency).*** Given two adjacent directed edges $e_1 = (u, v)$ and $e_2 = (v, w)$, there is a causality dependency between them, denoted by $e_1 \to e_2$, if and only if $f_e(e_1) < f_e(e_2)$, where $f_e$ extracts `starttime` of an event. Dependency is also defined for non-adjacent edges by transitivity: if $e_1 \to e_2$ and $e_2 \to e_3$, then $e_1 \to e_3$.

To conduct *causality analysis*, the analyst issues a query specifying the constraints to find the POI (Point-of-Interest) vertex $v$. The set of edges directly linked to $v$ (termed $E_v$) and the ones with causality dependency to $E_v$ will be returned. To notice, both forward-tracking (i.e., finding $e_{fwd}$ such that $e \to e_{fwd}$) and back-tracking (i.e., finding $e_{bck}$ such that $e_{bck} \to e$) can be supported, and in this work we focus on back-tracking [60], which is a more popular choice. Usually, newly discovered vertices/edges are returned to the analyst in an iterative way. The process will terminate when no more vertices/edges are discovered or the maximum

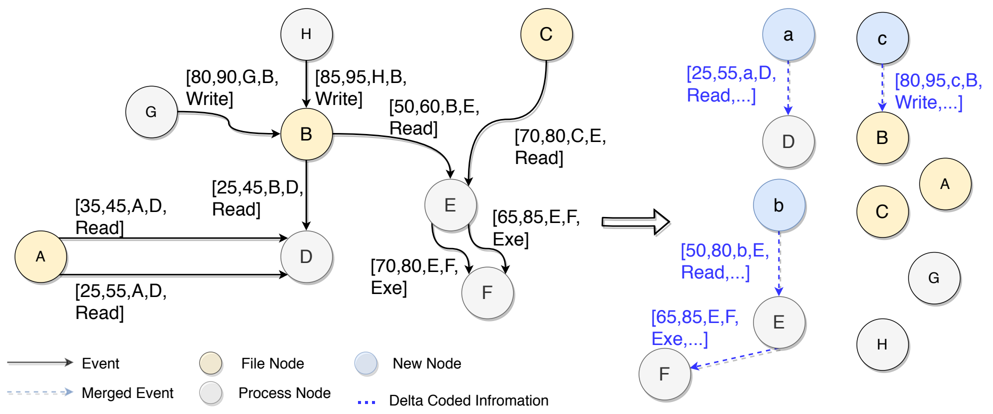


Figure 2.3: An example of dependency graph (left) and its compressed version after applying `SEAL` (right). Edges are merged if and only if they share the same destination node. To facilitate causality queries, the smallest `starttime` and the largest `endtime` are defined as the first two fields in the new edge. The ellipsis mark represents the time for all the compressed edges. For example, the edge between `a` and `D` is `[25,55,a,D,Read,(25,55),(35,45);(25,45)]`. We use comma to separate repeated edges and use semicolon to separate different edges. Therefore, combining with the node map in Table 2.2, we can see the compression is lossless. The edge properties will be further compressed as described in Section 2.3.4.

| New Node | Represented Nodes |
|---|---|
| a | A, B |
| b | B, C |
| c | G, H |

Table 2.2: Node map for the example in Figure 2.3.

depth specified by the analyst has been reached. In each iteration, the analyst can refine the query constraints to reduce the analysis scope.

Figure 2.3 (left) shows an example of a dependency graph generated from `FileEvent`. There are three file nodes (`A`, `B` and `C`) and five process nodes (`D`, `E`, `F`, `G`, `H`). The edge is formatted as [`starttime`, `endtime`, `srcid`, `dstid`, `accessright`]. Given a back-tracking query about POI vertex `F` and `starttime` ranged in [`45`, `100`], three causal events will be reported: [`70`, `80`, `E`, `F`, `Execute`], [`65`, `85`, `E`, `F`, `Execute`], and [`50`, `60`, `B`, `E`, `Read`]. The other edges do not satisfy the definition of causality analysis.

### 2.3.2 Query-friendly Compression

While compression is a well-developed area, with numerous methods available, many of them will introduce prominent overhead to causality analysis, as they require decompression every time a vertex/edge is examined. In this work, we adopt a concept from the data-mining community, termed *Query-friendly Compression (`QFC`)* [78, 31, 71], and develop compression techniques around it. In essence, the techniques under `QFC` should compress graphs "in a way that they still can be queried efficiently without decompression" [78]. For example, 4 types of queries can be supported with `QFC` algorithms of [31], including neighborhood queries, reachability queries, path queries, and graph pattern queries. Causality analysis can be considered as an iterative version of neighborhood queries.

Yet, the `QFC` schema of prior works cannot be directly applied to our setting. Firstly, some mechanisms require significant change on the data structures [78]. For our deployment, regular SQL queries have to be supported as well so the data format after compression has to adhere to the database schema. Secondly, the edges in all prior works have no associated properties [78, 31, 71], therefore only merging vertices is sufficient to fulfill their goal. While we can follow the same approach and keep the edge properties concatenated without compression, such a design is not optimal. Moreover, the queries on dependency graphs depend on not only the connectivity of the nodes but also the edge properties like `starttime`, leading to the challenge of retrieving the answers.

Therefore, we modify `QFC` according to causality analysis, which enforces *"decompression-free"* compression on graph structure and *"query-able"* compression on edge properties. Below we define the adjusted `QFC` based on the definition from [31].

**Definition 2 (*Query-friendly Compression*).** Assume a dependency graph $G = (V, E)$ is to be compressed. Let the class of causality analysis queries be $\mathcal{Q}$, and let $Q(G)$ be the answer to the query $Q \in \mathcal{Q}$. A `QFC` mechanism is a triple $< R, F, P >$, where $R$ is a compression method, $F : \mathcal{Q} \to \mathcal{Q}$ re-writes $\mathcal{Q}$ to accommodate the compressed data, and $P$ is a post-processing function. Compression can be expressed as $R(G) = R_p(R_s(G))$, where $R_s$ compresses the structures (vertices and edges), and $R_p$ compresses the edge properties or fields. Denote by $G_r = R(G) = (V_r, E_r)$ the graph after compression, such that $|E_r| \leq |E|$. `QFC` requires that for any query $Q \in \mathcal{Q}$,

• $Q(G) = P(Q'(G_r))$, where $Q' = F(Q)$ is the query on the compressed graph, and $P(Q'(G_r))$ is the result after post-processing the query answer on $G_r$.

• With only $R_s$ applied, any algorithm for evaluating $\mathcal{Q}$ can be directly used to compute $Q'(G_r)$ *without decompression.*

• When both $R_s$ and $R_p$ are applied, decompression is needed only when the relationship between the timestamps of a compressed edge $e \in E_r$ and the time range of the query cannot be determined.

Next, we describe our choices of $R_s$ and $R_p$ in Section 2.3.3 and Section 2.3.4. The query transformation $F$ and the post-processing $P$ are investigated in Section 2.3.5. Figure 2.3 (right) overviews the graph compressed with `SEAL`.

### 2.3.3 Compression on Graph Structure

We design the function $R_s$ such that multiple edges (from one pair of nodes or multiple pairs) can be reduced into a single edge. In particular, our algorithm finds sets of edges satisfying a certain *merge pattern* and combines all edges in the set. By examining the fields of `FileEvent`, one expects a higher compression ratio if edges with common fields are merged. Moreover, edges within proximity can be merged without sacrificing causality tracking performance. As illustrated in Figure 2.3, we choose the merge pattern to be *the set of all incoming edges of any node $v \in V$, which will share properties such as* ***`dstid`*** *or* ***agentid***. Correspondingly, a new node is added in the new graph $G_r$, representing the combination of all the parent nodes of $v$, if the number of parent nodes is more than 1.

We give an example in Figure 2.3. The new node `a` is generated to correspond to two individual nodes {`A,B`}, and the new edge [`25, 55, a, D, Read, (25, 55), (35, 45); (25, 45)`] is generated to correspond to three individual edges {[`25, 55, A, D, Read`], [`35, 45, A, D, Read`], [`25, 45, B, D, Read`]}. Similarly, we merge the two incoming edges of node `B`, merge the two incoming edges of node `E`, and create new nodes `c, b`, respectively. We also merge the two repeated edges between nodes `E, F`, but no new node needs to be created for them. Individual edges are removed in the compressed graph $G_r$ if they are merged into a new edge. However, as can be seen in Figure 2.3, individual nodes should not be removed. For example, even if the individual node `B` is included in the new node `a`, it cannot be removed because of its own incoming edges. The new nodes are recorded in a *node map*, shown in Table 2.2.

Our algorithm for $R_s$ is shown in Algorithm 1. It takes all the events as input, and creates two hash maps: (i) $NodeMap$, child node with all its parent nodes, and (ii) $EdgeMap$, a pair of nodes with all its corresponding edges. Then for each child node $v \in V$, all its parent nodes and the corresponding incoming edges are identified and merged. Meanwhile, the node map as in Table 2.2 is also updated. The time complexity of this algorithm is linear in the size of the graph, namely, $O(|V| + |E|)$. When responding to queries, decompression is selectively applied to *restore* the provenance, with the help of $NodeMap$ and $EdgeMap$.

### 2.3.4 Compression on Edge Properties

For all the properties or fields for a merged edge, they should be combined and compressed due to the redundant information, which is the focus of the compression function $R_p$. We propose delta coding for merged timestamp sequence, and Golomb code for the initial value

**Algorithm 1** Graph structure compression.

```
    Input: a set of edges $E$.
    Output: a set of new edges $E'$, a node map $NodeMap$.
 1: $NodeMap \leftarrow \emptyset$          ▷ hash map (key = a node, value = parent nodes)
 2: $EdgeMap \leftarrow \emptyset$          ▷ hash map (key = a pair of nodes, value = edges)
 3: for $e = (u, v) \in E$ do
 4:     $NodeMap$.put$(v, u)$
 5:     $EdgeMaps$.put$((u, v), e)$
 6: end for
 7: $E' \leftarrow \emptyset$
 8: for $v \in NodeMap$.keys do
 9:     $e' = \emptyset$                     ▷ a new edge
10:     $U \leftarrow NodeMap$.get$(v)$
11:     for $u \in U$ do
12:         $e' \leftarrow e' \cup \{EdgeMap$.get$((u, v))\}$
13:     end for
14:     $E' \leftarrow E \cup \{e'\}$
15: end for
```

in the sequence.

**Delta coding.** Delta coding represents a sequence of values with the differences (or delta). It has been used in updating webpages, copying files online backup, code version control and etc. [81]. We apply delta coding on timestamp fields (`starttime`, `endtime`) , as they usually share a long prefix. For instance, as shown in Figure 2.4, the `starttime` field is a long integer, and merged individual edges have values like 1562734588980, 1562734588971, 1562734588984, 1562734588990. Those values usually share the same prefix as the events to be compressed are often collected in a small time window, hence delta coding can result in a compact representation.

As shown in Figure 2.4, assume a node $x$ has $d$ incoming edges and $p$ parent nodes, $1 \leq p \leq d$. Let the `starttime` of the $j$-th edge be $t^j_{start}$, $1 \leq j \leq d$. We first construct a sequence

$$\mathbf{t}_{start} = [t^0_{start}; \quad t^1_{start}; \quad t^2_{start}, t^3_{start}; \quad ...; \quad t^d_{start} \, :]$$

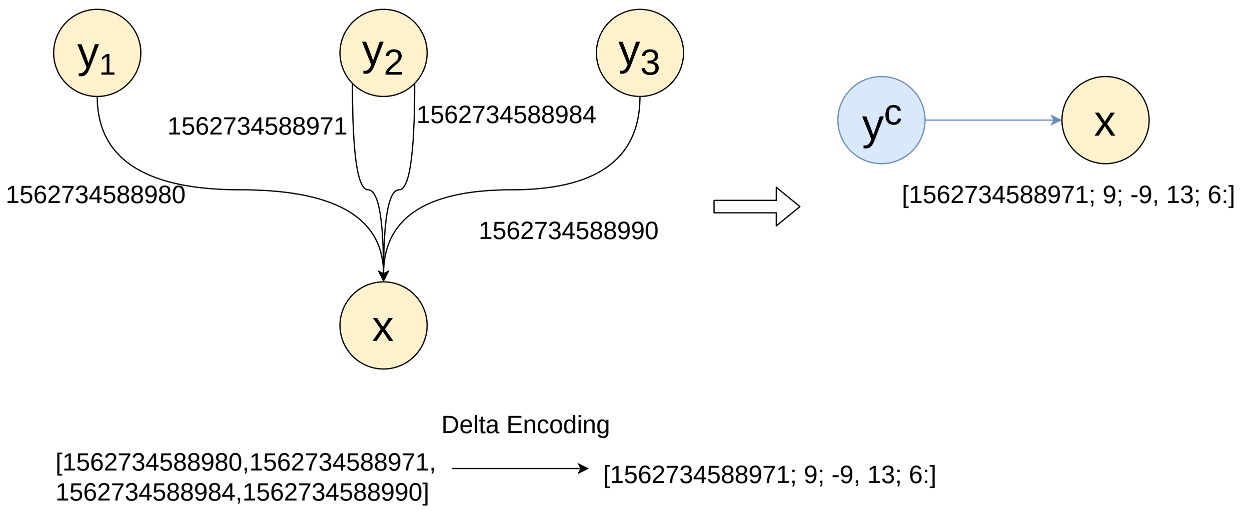


Figure 2.4: Delta coding for `starttime`. The first number in the combined time vector is the minimum time among the edges.

where $t^0_{start} = \min_{1 \le j \le d}(t^j_{start})$. Here comma is used to separate different edges from the same parent node, and semicolon separates different parent nodes. The colon at the end is used to separate the timestamp fields. For `endtime`, we choose the initial entry $t^0$ to be the *maximum* among the edges. Then we concatenate both fields into one sequence.

Then, we compute the delta for every consecutive pair of timestamps: for $1 \le j \le d$, $\Delta^j_{start} = t^j_{start} - t^{j-1}_{start}$. The resulting coded timestamp of the merged edge is:

$$[t^0_{start}; \quad \Delta^1_{start}; \quad \Delta^2_{start}, \Delta^3_{start}; \quad \Delta^4_{start}; \quad ..., \quad \Delta^d_{start} :]$$

and delta coding is also applied to the other timestamp fields. The time complexity of delta coding is $O(d)$ where $d$ is the number of edges.

To conform to the uncompressed `FileEvent` format, the $t^0_{start}$ and $t^0_{end}$ are stored in the `starttime` and `endtime` field of the new edge $e^c$ respectively, and the generated delta-coded `starttime` and `endtime` are stored in a new `delta` field.

**Golomb coding.** Delta coding can compress all the elements of the combined time sequence except $t^0$ which is still a long integer. Moreover, if an individual edge is not merged, its timestamps are also long integers. We choose to employ Golomb coding [45] to compress long integers to relatively small integers. Alternatively, a more aggressive approach is to use delta coding to compress $t^0$ of different merged events, but the database index will be updated [23] and the query cost will be high. One favorable property of Golomb coding is that the relative order of the numbers is not changed, which fits well with the requirements of `QFC`. That is, if $t > t'$, then we have the Golomb coded variable $\text{Gol}(t) > \text{Gol}(t')$.

Golomb code uses a parameter $M$ to divide an input datum $N$ into two parts (quotient $q$ and reminder $r$) by

$$q = \lfloor \frac{N-1}{M} \rfloor, \; r = N - qM - 1. \tag{2.1}$$

Under the standard Golomb coding schema, the quotient $q$ is then coded under unary coding, and the reminder $r$ is coded under truncated binary encoding to guarantee that the value after coding (called codeword) is a prefix code. In our case, however, the truncated binary encoding is not necessary because the codewords are separated by different entries automatically. As such we use a simpler mechanism, binary coding, for $r$. The coded data is then calculated by concatenating $p$ and $r$. For instance, given a long integer 1562734588980 (64 bits) and a $M = 1562700000000$, the binary form of $p$ and $r$ after coding will be 10 (2 bits) and 10000011111100100100110100 (26 bits). In this example, 32 bits are sufficient to store the Golomb codeword.

### 2.3.5 Query and Decompression

As defined by `QFC`, decompression is only necessary when the relation between the time range specified in the query and in the edge cannot be determined. If there are no intersections of these two ranges, decompression can be skipped. In our back-tracking queries, the above property holds for two reasons. First, due to the *order preservation* property of Golomb coding, it is unnecessary to decode all Golomb codes in the database to answer a query with a timestamp constraint. The specified timestamp can be simply encoded by Golomb code, and used as the new constraint issued to the database. Second, the minimum `starttime` $t^0_{start}$ is recorded in a merged edge. Hence, if we back-track for events whose `starttime` is smaller than some given $t_{query}$, then all individual edges of an combined edge with $t^0_{start} > t_{query}$ will be rejected. Therefore, the database does not need to decompress and can safely reject this combined edge.

Here we use the example shown in Figure 2.3 to demonstrate how the query and decompression work. Assume a query tries to initiate back-tracking on `E` to find the prior causal events whose `starttime` is less than $t_{query} = 65$. First, $t_{query}$ will be Golomb coded into $Gol(65)$. And the database needs to find events such that $Gol(t^0_{start}) < Gol(65)$ and the destination node is `E`. For the merged event [`50, 80, b, E, Read`], its $t^0_{start} = 50$ value is stored as $Gol(50)$. By order preservation of Golomb code, $Gol(50) < Gol(65)$. Thus this merged event will be identified. Second, we decompress this merged event for further inspection. We extract `starttime` $Gol(t^0_{start}) = Gol(50)$ and Golomb decoding is applied to obtain $t^0_{start} = 50$. Then we recover the timestamp sequence $\mathtt{t}_{start}$ by calculating $t^j_{start} = t^{j-1}_{start} + \Delta^j_{start}, j \geq 1$. In this example, $t^1_{start} = 50, t^2_{start} = 70$. Comparing the individual timestamps now is feasible. It will be found that only the first individual edge is a valid answer. The final step is to find the individual nodes corresponding to the valid edges from the node map in Table 2.2. After that, the result [`50, 60, B, E, Read`] is returned.

It can be seen that if $t_{query} = 30$, all incoming edges of `E` can be rejected without Golomb or delta-code decompression ($t_{query}$ still needs to be Golomb encoded before issuing the query).

### 2.3.6 Compression Ratio Estimation

Applying compression to the log data may be desirable only if the compression ratio is higher than a threshold. As a result, it is important to obtain the compression ratio or its estimate before compression. While a full scan of the causality graph gives a precise compression ratio, the overhead is significant. As a result, we develop an algorithm to estimate the compression ratio. To that end, we show that this estimation is reduced to obtaining $d_{avg}$, the average degree of the undirected version of the causality graph (Appendix 6.4). A degree estimator is developed with a sample size only depending on the required accuracy rather than on the number of nodes or the number of edges. As described in Section 2.4, we implement `SEAL` for online compression, this algorithm is applied to chunks of data sequentially.

**Compression ratio estimation.** Let $G_{undirected}$ denote the undirected graph which is identical to the dependency graph except that edge directions are removed. Let $d_{avg}$ be its average node degree. From Appendix 6.4, we find that the compression ratio is an explicit function of $d_{avg}$. The compression ratio estimation reduces to estimating the average degree. To minimize the data access and query time during estimation, we present an average degree estimation algorithm that samples nodes in an undirected graph $H$ based on random walk (see Algorithm 2). The algorithm can be applied to $H = G_{undirected}$ and outputs $d_{avg}$. In the following, we use the notation $d_H$ for the average degree of $H$, and $\hat{d}$ the estimated average degree. For any vertex $v$ of $H$, denote by $d_v$ its degree.

One way to estimate the average degree is to uniformly sample nodes in $H$ and get their degrees, and obtain the average of the sampled degrees [32]. The estimator from the sample set $S$ is:

$$\hat{d} = \frac{\sum_{v \in S} d_v}{|S|} = \frac{\sum_{v \in S} d_v}{\sum_{v \in S} 1}. \tag{2.2}$$

This method can be improved when we also obtain a random neighbor of each sampled nodes

[44]. The required number of samples (sample complexity) is $O(\sqrt{n})$ to obtain a constant-factor estimation, where $n$ is the number of nodes. Another way is to sample nodes according to the node degree, and use collisions in the samples to obtain the estimate [57], where the required sample complexity is $\Omega(\sqrt{n})$. Our algorithm is inspired by the 'Smooth' algorithm of [26], where a node $v$ is sampled proportional to its degree plus a constant, $d_v + c$, where the constant $c = \alpha d_H$ is a coarse estimate of the average degree with a multiplicative gap $\alpha$. The coarse estimation $c$ can be obtained from history or a very small subgraph in our problem. The resultant sample complexity is no more than $\max(\alpha, \frac{1}{\alpha})\frac{6}{\epsilon^2}\log\frac{4}{\delta}$, and the average degree estimate $\hat{d}$ satisfies

$$Pr\Big((1-4\epsilon)d_H \leq \hat{d} \leq (1+4\epsilon)d_H\Big) \geq 1-\delta, \tag{2.3}$$

for all $0 < \epsilon \leq 0.5, 0 < \delta < 1, \alpha > 0$.

In large graphs, it is hard to sample nodes in the entire graph according to some distribution as we do not know the number of nodes and the node degrees. To overcome such difficulty, the Smooth algorithm can be modified such that the sampled nodes are obtained by random walk [26]. However, it makes some assumptions that do not readily fit the dependency graph problem: (i) The graph needs to be irreducible and aperiodic. However, the dependency graph naturally contains disconnected components. (ii) The sample complexity needs to be high enough to pass the mixing time and approach the steady-state distribution, which varies depending on the structure of the graph.

To overcome these issues, two techniques are used in Algorithm 2. First, random walk with escaping [5] jumps to a random new node with probability $p_{jump}$ and stays on the random walk path with probability $1 - p_{jump}$ (see Line 4). Therefore, we can reach different components of the graph. Second, thinning [57] takes one sample every $\theta$ samples as in Line 7. We obtain $\theta$ groups of thinned samples. If the samples are indexed by $0, 1, 2, \ldots,$ then in our algorithm the $j$-th group, denoted by $S_j$, contains samples indexed by $j, j+\theta, j+2\theta, \ldots$, for $0 \leq j \leq \theta - 1$. Each group produces its own estimate (Line 13), and the final estimate is the average of these groups (Line 14). Since the sample distribution is not uniform, we cannot directly use the estimator of Equation (2.2). The sampled degrees need to be re-weighted using the Hansen-Hurwitz technique [49] to correct the bias towards the high degree nodes, corresponding to the term $d_v + c$ in the numerator and the denominator of Line 13. Note that due to the difficulty to sample a node from the entire graph, the sample distribution is not specified in Lines 2 and 9.

**Algorithm 2** Average degree estimation.

```
    Input: undirected graph $H$, sample size $r$, coarse average degree estimator $c$, thinning
    parameter $\theta$, jumping probability $p_{jump}$
    Output: average degree estimator $\hat{d}$
 1: $S_j \leftarrow \emptyset, j = 0, 1, \ldots, \theta - 1$
 2: Randomly sample a node $v_{pre}$ of $H$
 3: for $i = 0$ to $r - 1$ do
 4:     $rnd \sim \text{Bernoulli}(p_{jump})$
 5:     if $rnd = 0$ then
 6:         Uniformly sample a neighbor $v$ of $v_{pre}$ assuming $v_{pre}$ also has $c$ added self loops
 7:         $S_{i \bmod \theta} \leftarrow S_{i \bmod \theta} \cup \{v\}$
 8:     else
 9:         Randomly sample a node $v$ of $H$
10:     end if
11:     $v_{pre} \leftarrow v$
12: end for
13: $\hat{d}_j = \frac{\sum_{v \in S_j} d_v/(d_v+c)}{\sum_{v \in S_j} 1/(d_v+c)}, j = 0, 1, \ldots, \theta - 1$
14: $\hat{d} = \frac{1}{\theta} \sum_{j=0}^{\theta-1} \hat{d}_j$
```

## 2.4 Architecture

**Design rationale.** Figure 2.5 shows the architecture of `SEAL` and how it is integrated into the log ingestion and analysis pipeline. `SEAL` resembles the design [111] at the very high level. In [111], the compression system mainly includes three elements: computing components, caches, and the database. In this work, we redesign those elements according to our algorithm for both the compression system and the query system. The *compression system* receives online data streams of system events, encodes the data, and saves them into the database. The *query system* takes a query, applies the query transformation and recovers the result with post-processing, and returns the result. The information flow follows closely the definition of QFC in Section 2.3.2 and includes the structure and property compression $R_s, R_p$, the query transformation $F$, and the post-processing $P$, which are explained in details in Sections 2.3.3 – 2.3.5.

Due to the current monitoring system structure of our industrial collaborator, SEAL is solely deployed at the server-side by the data aggregator. Note that, alternatively, one can choose to compress the data at the host end before sending them to the data aggregator. Since there are no cross-host events in FileEvent, the compression ratio will be identical for both choices.

**Online compression.** While offline compression can achieve an optimized compression ratio with full visibility to the data, it will add a long waiting time before a query can be processed. Given that causality analysis could be requested any time of the day, offline compression is not a viable option. As such, we choose to apply online compression.

The online compression system is built by the following main components: (i) The optional compression ratio estimator. If the estimated ratio as described in Section 2.3.6 is more than the given threshold, data is passed through the following components. Otherwise, data is directly stored in the database. (ii) Caching. It organizes and puts the most recent data stream into a cache. When the cache is filled, the data will be compressed. The cache size is configurable, called *chunk size*. (iii) Graph structure compression. It merges and encodes all the edges that satisfy the edge merge pattern as in Section 2.3.3. It also generates the node mapping between the individual nodes and the new nodes, shown in Table 2.2. (iv) Edge property compression. It encodes each event timestamp entry using delta coding and

Golomb codes as in Section 2.3.4.

Next, we remark on some design choices. The configurable chunk size provides a tradeoff between the memory cost and the compression ratio. The larger the chunk size, the more edges can be combined. We found in our experiments as in Section 2.5 that 134 MB per host is a large enough chunk size offering sufficiently high compression capability.

**Query.** The query system comprises three main components. (i) Query transformation. Given a query $Q$, `SEAL` transforms it into another query $Q'$ that the compressed database can process. In particular, it needs to transform the queried timestamp and the `srcid` constraints, if there are any. The timestamp constraint is encoded into a Golomb codeword, which is used as the new constraint as in Section 2.3.5. If a `srcid` is given, then this individual node is mapped to all the corresponding new nodes using the node map. (ii) Querying. The transformed query $Q'$ is issued to the database and the answer is obtained. (iii) Post-processing. The combined edges are decompressed from delta codes, the timestamp constraint is checked, the merged node is mapped to individual nodes, and the valid individual edges are returned as described in Section 2.3.5.

Note that, in the query transformation component, if `dstid` is a query constraint, then no node mapping is required since only source nodes are merged during compression. In our work, we focus on back-tracking, where `srcid` is not a query constraint, hence the query transformation is simplified. Moreover, the node mapping progress is fast due to the small number of objects compared to the events.

For each given destination ID, at most one combined edge will be returned as an answer in each chunk (containing $10^5$ to $10^6$ events depending on the chunk size). This observation combined with the fact that the dependency graph is much smaller after compression effectively controls the query overhead in our experiments.

To quickly access the node map as in Table 2.2, it is cached using a hash map. Given that

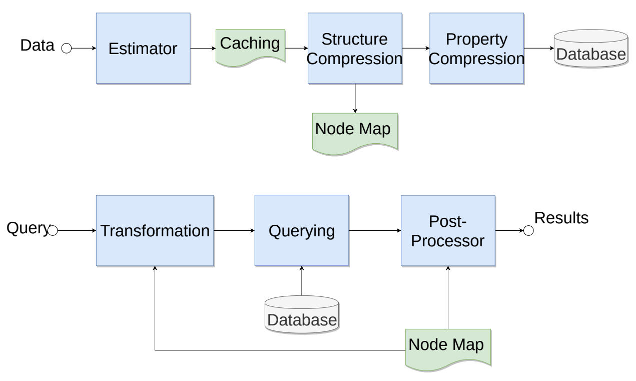


Figure 2.5: The SEAL architecture of online compression and querying.

the number of nodes is much smaller than the number of edges, the memory size of this hash map is a small fraction of the database size.

## 2.5 Evaluation

### 2.5.1 Experiment Setup

Our evaluation about compression is primarily on a dataset of system logs collected from 95 hosts by our industrial partner, which we call $DS_{ind}$. This dataset contains 53,172,439 events and takes 20GB in uncompressed form. For querying evaluations, we select a subset of $DS_{ind}$ covering 8 hosts, with 46,308 events and a total size of 8 GB. As $DS_{ind}$ does not have ground-truth labels of attacks, we use another data source under the DARPA Transparent Computing program [25]. The logs are collected on machines with OS instrumented, and a red team carried out simulated APT attacks. Multiple datasets are contained, and each one corresponds to a simulated attack. We use CARDETs dataset, which simulates an attack on Ngnix server, with a total of 1,183M events (27% write, 25.8% read and 47.2% execute), and we term this dataset $DS_{dtc}$. Since our system focuses on event merging, we only compress the edges and a subset of the attributes, with 233GB data size.

We implemented `SEAL` using JAVA version 11.0.3. We use JDBC (the Java Database Connectivity) to connect to PostgreSQL Database version Ubuntu 11.3-1.pgdg18.04+1. For $DS_{ind}$, we run our system on Ubuntu 14.04.2, with 64 GB memory and Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHZ. To run the queries, one machine with AMD Ryzen 7 2700X Eight-Core Processor and 16GB memory is used. For $DS_{dtc}$, we run the system on Ubuntu 16.04, with 32 GB memory and Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz.

Section 2.2.3 compares the designs between `SEAL` and other systems, and demonstrates when other systems introduce errors to attack investigation. In this section, we quantify the difference, and select the method of Full Dependency (FD) preservation [52] as the comparison target, which strikes a good balance between reduction rate and preservation of analysis results. Under FD, A node $u$ is *reachable* to $v$ if either there is an edge $e = (u, v)$ or there is a causality dependency $e_u \rightarrow e_v$, where $e_u$ is an outgoing edge of $u$, and $e_v$ is an incoming edge of $v$. We implement a relaxed FD constraint, where repeated edges (between any pair nodes) are merged such that the reachability for any pair of nodes in the graph is maintained. The corresponding compression ratio is better than FD since it is a relaxation. We compare the relaxed FD with our method `SEAL`.

Our evaluation focuses on the following aspects. In Section 2.5.2, we study the data compression ratio and the number of reduced events for different hosts and different `accessright` operations (read, write, and execute). We demonstrate the impact of the assigned chunk size (for caching events) on the reduction factor. We compare our method to relaxed FD on compression rate. In Section 2.5.3, we compare the processing time of running backtracking queries on the compressed and uncompressed databases. For the compressed case, the time for the database to return the potential merged events and the time for `SEAL` to post-process them are investigated. We show the accuracy advantages of lossless compression under queries with time constraints. Finally in Appendix 6.4, we evaluate the accuracy of the average degree estimator and compare it with direct uniform sampling.

| Host ID | Event Count / Reduction | Read % / Reduction | Write /Reduction | Execute / Reductio |
|---|---|---|---|---|
| 5 | 278913 / 9.25x / 5.85x | 61% / 6.6x / 4.1x | 11% / 9.2x / 8.4x | 28% / 65.3x / 33.0 |
| 23 | 880162 / 25.45x / 19.14x | 91% / 37.7x / 26.9x | 8% / 5.3x / 4.5x | 1% / 35.8x / 13.2x |
| 52 | 523671 / 41.45x / 17.36x | 70% / 39.1x / 14.8x | 22% / 54.9x / 47.5x | 8% / 36.6x / 14.7x |
| 3 | 312392 / 15.37x / 13.31x | 36% / 12.6x / 10.8x | 29% / 8.8x / 8.4x | 34%/ 125.8x / 52.3 |
| 94 | 517978 / 78.82x / 26.9x | 20% / 19.8x / 6.1x | 8% / 200.6x / 96.3x | 72% / 346.0x / 209. |
| All | 53172439 / 9.81x / 5.71x | 65% / 10.3x / 5.5x | 19% / 5.3x / 3.7x | 15% / 76.3x / 38.7 |

Table 2.3: Example hosts and the reduction factors. The reduction factors are measured for two chunk sizes: $10^6$ and $10^5$. The last row shows the overall result for the 95 hosts.

### 2.5.2 Compression Evaluation

**Compression ratio.** We measure the compression ratio as the original data system over the compressed data system using the above two chunk sizes. For $DS_{ind}$, when the chunk size (number of cached events) is $10^6$, the compressed data is reduced to 7.6 GB from 20 GB, resulting in a compression ratio of **2.63x**. For $DS_{tpc}$, the chunk size equals one file size and contains around $5 \times 10^6$ events. The compressed size is 18 GB reduced from 233GB, resulting in a compression ratio of **12.94x**.

**Reduction factor for different operations and hosts.** To further understand the compression results, we investigate the reduction factor, defined as the number of original events divided by the number of compressed events. We focus on $DS_{ind}$, and some examples of the hosts and the average reduction factors from 95 hosts are illustrated in Table 4.2. In the table, we list results for the chunk size of $10^6$ as well as $10^5$.

It can be observed that the types of events (read, write, and execute) differ by the hosts. We observed that in $DS_{ind}$, read is the most popular operation among most of the hosts, where 72 hosts have more than 50% read events. Write is much less prevalent in general, where 67 hosts have between 10% to 30% write events. Finally, execution varies by the host, and 69 hosts have between 10% to 60% executions.

On average, the reduction factor of execute events is higher than reads, and writes have the

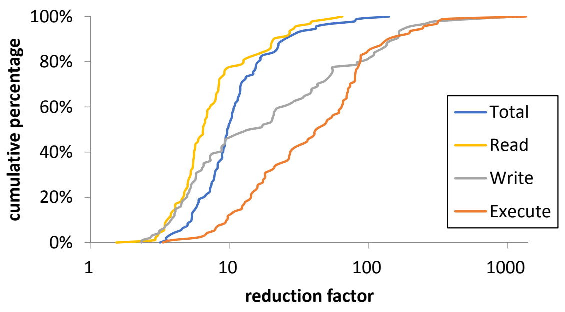


Figure 2.6: The cumulative distribution of the reduction factors for the 95 hosts in $DS_{ind}$. The reduction factor is calculated for all three types of operations, read, write, and execute, and the overall events in each host.

lowest reduction factor, as can be seen from the last row of Table 4.2. However, for each host, this ordering changes depending on the structure of the dependency graph, e.g., if there exist many repeated events between two nodes. Host 5 is an example that has reduction factors similar to the average case. Hosts 23, 52, 3 see higher reduction factors of read, write, and execute events, respectively. Host 94 is an example of high reduction factors for all events. In Figure 2.6 we plot the cumulative distribution of the reduction factors among the 95 hosts.

The number of events of a host affects the reduction factor to some extent. In particular, if the number of events is less than the chunk size, as occurred for a few hosts when the chunk size is $10^6$, the cache is not fully utilized, and fewer merge patterns may be found. However, some hosts with a small number of events still outperform the overall case as the last row in Table 4.2, due to their high average degree.

Previous works like NodeMerge focus on one type of operation, such as read [111], and show a high data reduction ratio on their dataset. Our result suggests such an approach is not always effective, when compressing data from different types of machines (e.g., Host 94). As such, `SEAL` is more versatile to different enterprise settings.

**Chunk size.** When the chunk size is increased from $10^5$ to $10^6$, the overall reduction factor is increased by 1.7 as in the last row of Table 4.2. Correspondingly, the consumed memory

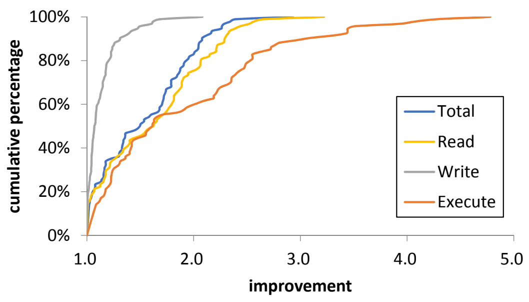


Figure 2.7: The cumulative distribution of the improvement over the 95 hosts when the chunk size is increased from $10^5$ to $10^6$. The improvement for Read, write, execute, and overall events in each host is calculated.

size is increased from 134 MB to 866 MB. The cumulative distribution of the reduction improvement, which is the reduction factor of chunk size $10^6$ divided by that of chunk size $10^5$, is plotted in Figure 2.7. The improvement is due to the fact that when more events are considered in one chunk, more edges exist in the dependency graph, but the number of nodes does not increase as fast. A larger average degree and hence a larger reduction factor is achieved. It can be seen that the execute events change the most with a larger chunk size, while the write events change the least with the chunk size. This also is consistent with the fact that executions have more repeated edges between processes while write events operate on different files over time.

**Comparison to FD.** We use $DS_{dtc}$ to compare `SEAL` and FD, as the DARPA data is also used by Hossain et al. [52]. Figure 2.8 shows the compression ratio of four methods: 1) "optimal" – keeping only one random edge between any pair of nodes, which violates causality dependency but gives an upper bound on the highest possible compression ratio when repeated edges are reduced, 2) "FD" – removing repeated edges under relaxed full dependency preservation, 3) "SEAL repeat edge" – our method that only compresses all repeated edges, and 4) "SEAL" – our method that compresses all incoming edges of any node.

Figure 2.8 shows that if we only compress the repeated edges by SEAL, we can get almost the same compression ratio as FD. Both methods are close to the minimum possible compressed

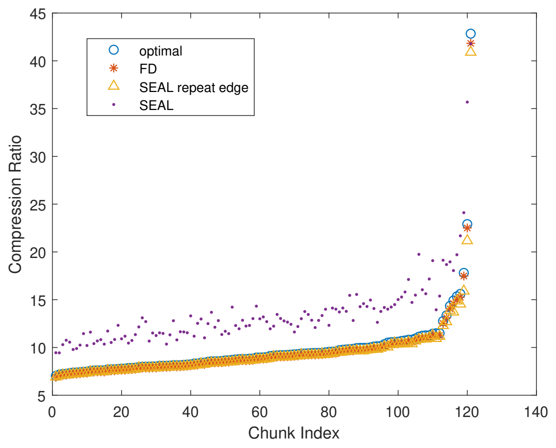


Figure 2.8: Comparison between our methods and FD.

size under repeated edge compression. Besides, if we compress all the possible edges using `SEAL`, we get a compression ratio of 12.94x compared to 8.96x for FD preservation.

### 2.5.3 Query Evaluation

We measured the querying and decoding time cost of `SEAL` as well as the querying time of the uncompressed data. We use a dataset with 830,235 events under $DS_{ind}$ and run back-tracking through breadth-first search (BFS) to perform the causality analysis for every node. We use BFS here as it can be seen as a generalization of causality analysis: if no additional constraints are assumed, causality analysis is BFS under causality dependency. In particular, starting from any POI node $x$, we query for all incoming edges $e_1, e_2, \ldots, e_d$ and the corresponding parent nodes $y_1, y_2, \ldots, y_d$, where $d$ is the incoming degree of $x$. Then for each node $y_i$, $1 \leq i \leq d$, we query for its incoming events whose `starttime` is earlier than that of $e_i$. The process continues until no more incoming edge is found.

Figure 2.9 shows the performance of this evaluation. The querying and the decoding time on the compressed data normalized by the querying time on the uncompressed data are plotted. We obtain 133 start nodes each of which returns more than 2,000 querying results. We observe that **89%** starts nodes (118 out of 133) use less time than the uncompressed

| NID | Number of Reachable Nodes/Edges | | | |
|---|---|---|---|---|
| | Uncmp | `SEAL` | Cnstrnd Uncmp | Cnstrnd `SEAL` |
| 1 | 1093/4302 | 1093/4302 | 293/779 | 293/779 |
| 2 | 9496/37944 | 9496/37944 | 1457/5999 | 1457/5999 |
| 3 | 178/616 | 178/616 | 116/358 | 116/358 |
| 4 | 45/3739 | 45/3739 | 11/2113 | 11/2113 |

Table 2.4: The results of back-tracking starting from 4 nodes. "NID", "Uncmp" and "Cnstrnd" are short for "Node ID", "Uncompressed" and "Constrained".

data, and 30 start nodes use less than half the time of the uncompressed data. Moreover, on average decompression only takes 18.66% of the overall time, because only potentially valid answers are decompressed. It is also observed that the querying time of `SEAL` is only 63.87% of the querying time for uncompressed data. For $DS_{dtc}$, `SEAL` runs on about 5.27M nodes, 15.47% nodes use less time than the uncompressed data, and on average takes 1.36x time of the uncompressed data.

Note that queries usually have a restrictive latency requirement while compression of collected logs can be performed at the background of a minoring server. Our method tradeoff the computation during compression for better storage efficiency and query speed.

**Evaluation of attack provenance.** Here we use the simulated attacks of $DS_{dtc}$ to evaluate whether `SEAL` preserves the accuracy for data provenance. We use four processes on two hosts (two for each) which are labeled as attack targets (`ta1-cadets-2` and `ta1-cadets-1`) as the starting nodes. Then we run the BFS queries, and count 1) the number of nodes reachable from a starting node (reachable is defined in Section 2.5.1) and 2) the number of edges from a starting node to all its reachable nodes. Table 2.4 (Columns 2 and 3) shows the number of reachable nodes and edges in the BFS graph. It turns out `SEAL` returns the exact same number of reachable nodes and edges as the uncompressed data, indicating it preserves provenance accuracy. Next, we demonstrate the versatility of our lossless method for queries with time constraints, for example, when the analyst knows that the attack occurred in

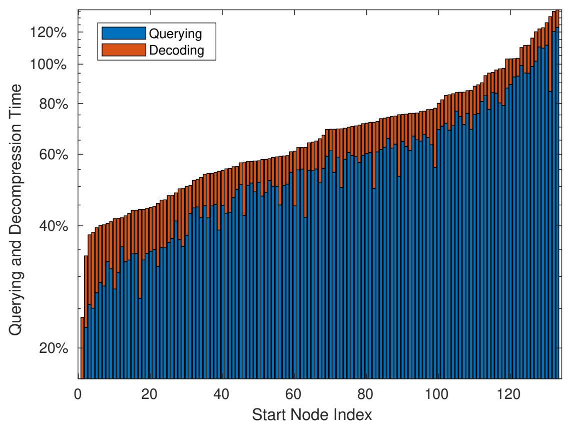


Figure 2.9: Querying and decompression time of back tracking with 133 start nodes that return the largest result sizes, normalized by the querying time of the uncompressed data. The nodes index are sorted by the query time.

an approximate time period $[t_1, t_2]$. Since our lossless compression can restore all the time information, we can add arbitrary constraints to our analysis without any concerns, which is verified by the last two columns (Column 4 and 5) of Table 2.4. Lossy reduction methods, such as FD, even though preserve certain dependency, still lose time information once edges are removed, and thus might introduce false connectivity under time constraints (see Figure 2.2 for an example).

## 2.6 Discussion

**Limitations and future works.** $DS_{ind}$ is collected for a small number of days and a subset of all hosts from our industrial partner. Therefore, a larger dataset may provide a more comprehensive understanding of the performance for `SEAL`. The compression ratio can be further improved through two possible methods. First, the proposed algorithms reduce the number of events, but the properties of all merged events are losslessly compressed together. Even though such compression produces a hundred percent accuracy for log analytics and the merge patterns can be easily found, dependency-preserving timestamp lossy compression may improve the storage size. Second, domain-specific knowledge can be explored such as removing temporary files [62]. Another limitation is the memory overhead to store the node map as in Table 2.2, which is the only extra data other than the events. Our experiment results show that the node map takes 114 MB on disk, but consumes 1.4 GB when loaded into memory. The memory cost can be reduced by replacing generic hash maps of Java with user-defined ones.

**Potential attacks.** When the adversary compromises end-hosts and back-end servers, she can pro-actively inject/change/delete events to impact the outcome of `SEAL`. Log integrity needs to be ensured against such attacks, and the existing approaches based on cryptography or trusted execution environment [11, 56, 102, 91, 83] can be integrated to this end.

One potential attack against `SEAL` is denial-of-service attack. Though delta coding and Golumb coding are applied to compress edges, all timestamps have to be "remembered" by the new edge. The adversary can trigger a large number of events to consume the storage. This issue is less of a concern for approaches based on data reduction, as those edges will be considered as repeated and get pruned. Moreover, knowing the algorithm of compression ratio estimation, the adversary can add/delete edges and nodes to mislead the estimation process to consider each block incompressible. On the other hand, such denial-of-service attack will make the performance of casualty analysis fall back to the situation when no compression is applied at most. The analysis accuracy will not be impacted. Besides, by adding/deleting an abnormal number of events, the attacker might expose herself to anomaly detection methods.

**Out-of-order logs.** Due to reasons like network congestion, logs occasionally arrive out of order at the back-end analysis server [125]. Since the dependency graph possesses temporal locality, such "out-of-order" logs result in potential impact on the compression ratio. This issue can be addressed by the method described as follows. Assuming the probability of out-of-order logs is $p$, the server can reserve $pN$ temporary storage to hold all out-of-order logs in a day, where $N$ is the daily uncompressed log size. During off-peak hours, the server can process each out-of-order log. For log from Node $u$ to Node $v$, we 1) retrieve in the compressed data the merged edges to $v$ and decompress the timestamps, and 2) merge the edge $(u, v)$ with the retrieved edges and compress the timestamps. Since the probability $p$ is typically small and off-peak hours are utilized, out-of-order logs can be handled with smoothly.

**Generalizing SEAL.** Though SEAL is designed for causality analysis in the log setting, it can be extended to other graphs/applications as well. Generally, SEAL assumes the edges of a graph have attributes of timestamp, and the application uses time range as a constraint to find time-dependent nodes/edges. Therefore, the data with timestamp and entity relations, like network logs, social network activities, and recommendations, could benefit from SEAL. Besides forensic analysis, other applications relying on data provenance, like fault localization, could be a good fit. We leave the exploration of the aforementioned data/applications as future work. In terms of the execution environment of SEAL, we assume SQL database stores the logs on a centralized server, like prior works [111, 122, 62, 52]. It is possible that the company deploying SEAL in a distributed environment (e.g., Apache Spark) with non-SQL-based storage. How to adjust SEAL to this new environment worth further research as well.

## 2.7 Conclusion

Causality analysis reconstructs information flow across different files, processes, and hosts to enable effective attack investigation and forensic analysis. However, it also requires a large amount of storage, which impedes its wide adoption by enterprises. Our work shows the concern about storage overhead can be eased by query-friendly compression. Comparing to prior works based on data reduction, our system SEAL offers similar or better storage (e.g., 9.81x event reduction and 2.63x database size reduction on $DS_{ind}$) and query efficiency (average query speed is 64% of the uncompressed form) with guarantee of no false positive and negative in casualty queries. We make the first attempt to integrating the techniques in the coding area (like Delta coding and Golumb coding) with a security application. We hope in the future more security applications can be benefited with techniques from the coding community and we will continue such investigation.

# Chapter 3

# LDPC Codes for DNA Storage with Nanopore Sequencing

With the fast growth of data, DNA storage attracts much attention for its extremely large data densities, integrity in non-ideal conditions over millennial, and efficient data replication (e.g., [21, 43, 124, 10, 123, 90, 70]). In DNA storage, the writing process is called *synthesis*, which joins neulitide symbols and produces the desired DNA string. Reading is completed through DNA *sequencing* that reads the string and translates it to digital data.

Among the DNA sequencing technologies, nanopore sequencing performed on the MinION device [121] is a promising one due to its low cost, scalability, and portability, and has been demonstrated suitable for DNA storage. In nanopore sequencing, a DNA sequence is passed through a nano-sized hole and current variation is measured to estimate the sequence symbols. While many proposed DNA storage systems and research use short strands of DNA to store information, the "third-generation" technologies including Oxford Nanopore[55] can provide reads with thousands of nucleotides. Short strands introduces high loss in coding efficiency and underutilized the nanopore sequencing [123]. Therefore, it can be suitable for long-codeword methods such as low-density parity-check (LDPC) code, which is the focus of our paper.

Data storage using nanopore sequencing was first demonstrated in [123]. To accommodate the use of nanopore sequencing, 17 identical long DNA fragments (1000 base pairs(bp)) encoding for a 3kB file were synthesized, and subsequently sequenced and decoded using ONT MinION platform. The work in [70], arguing that existing scalable approaches for synthesis rely on short oligonucleotides (i.e. 100 - 200 bases in length), stored 1.67 Mb file in 111.499 short oligonucleotides and used Gibson Assembly[41] to concatenate short oligonucleotides to large one (5000 bp) for sequencing reads.

Moreover, errors in nanopore sequencing [33] should be addressed in order to maintain data integrity. In [99] burst deletion errors are considered and associated codes are constructed. In [75] a channel model of nanopore sequencing is established capturing non-linear inter-symbol interference, deletions, and substitutions. In [22] the measured current is modeled as a quaternary amplitude modulation with additive white Gaussian noise. In [36] it is observed that substitution errors occur asymmetrically among the four bases of nucleotides $(A, T, G, C)$, and the minimum asymmetric Lee distance are studied. In [16] it is mentioned that most previous works rely heavily on a large number of mulitple reads which leads to a high reading cost. One strategy to deal with this problem is working with the the raw nanopore signal and hence the convolutional code[119] has been used in its work.

The goal of this work is to design LDPC codes for the asymmetric substitutions of nanopore sequencing. Instead of studying the minimum code distance as in [36], we focus on the performance in terms of bit error rate, and LDPC codes are chosen for its near-capacity performance, low encoding/decoding complexity and availability working with the soft information. As the DNA string is a sequence of alphabet size 4, a quaternary code is a natural candidate for error correction. Although the LDPC code supports non-binary alphabets, it suffers from slow speed in decoding compared to the binary case. Hence, we focus on binary

LDPC codes for DNA storage, and address the following questions: *How to utilize the special structure of the asymmetric channel to design binary decoding methods? Can such methods approach the fundamental limits of the channel?*

The key ideas in our techniques are that we can view each quaternary symbol as two bits, and the asymmetric channel introduces a Boolean constraint on the two bits with high probability given certain channel outputs. As a result, the two bits can exchange information during the decoding process and guide each other to reach the correct codeword symbol. The effectiveness of such decoding methods in the asymptotic regime of large block length can be verified using density evolution [97], however the challenge is to make proper modifications to accommodate the information exchange steps and the asymmetry of the channel.

The contributions of the paper are as below.

- We present an asymmetric nanopore channel model for both continuous/soft and discrete/hard channel output. The continuous output corresponds to the analog current signals measured for each nuceotide, and is modeled by 2-dimensional Gaussian random variables. The discrete output corresponds to maximum likelihood hard decision from the Gaussian output. The channel capacities are calculated for independent uniformly distributed (i.u.d.) inputs.

- We propose to use two binary LDPC codes for one quaternary DNA sequence, and show decoding algorithms that exchange information between the two LDPC codes. The information exchange step is inspired by Turbo codes, and addresses the asymmetry in the channel errors. We design three information exchange methods and also utilize a fourth decoding method proposed for multiple access channel [92].

- Density evolution algorithms are derived for the above decoding methods. In particular, two main challenges are considered: 1. We need to add exchange information for two density evolution corresponding to our decoding algorithms. 2. Density evolution assumes the all-zeros codeword is sent due to the symmetry condition. Our method, although satisfies the symmetry condition of density evolution, can not make the identical input assumption because it only passes the extrinsic information under a specific condition. For the information exchange issue, we design the density evolution algorithms correspondingly for the three information exchange methods. For the all-zeros codeword issue, we design algorithm to average the i.u.d inputs.

- We also consider multiple reads collected for the same codeword, a performance-boosting technique commonly seen in sequencing. When a codeword is read $M$ times, we model the channel output as $2M$-dimensional Gaussian under soft coding, where the multiple reads are assumed to be independent.

- Simulation shows that our method has a 20X speed-up compared to the quaternary codes and comparable bit error rates. Moreover, soft decoding has over $10^4$X lower bit error rate compared to the hard decoding in our methods. Also, the multiple reads provide over $10^4$X lower bit error rate compared to the original methods.

- Density evolution results show that our methods can approach the capacity of the Gaussian channel for nanopore sequencing with a gap of 0.0234. For the discrete hard decision channel, our method is also close to the i.u.d. capacity, with a gap of 0.0684.

**Related work.** Motivated by improving the reliability of DNA synthesis and sequencing, there has been quite a few research works on coding for DNA storage. The work in [110] presents an algorithm that maps digital data to three DNA nucleotides. The algorithm avoids the repeating identical nucleotide symbols. The authors in [8] developed a forward-error-correcting scheme that could be used to avoid multiple errors during the DNA synthesis. The work in [50, 104] writes data in many DNA sequences that are stored without order and investigates the corresponding channel capacity. It proves that the simple scheme of adding an index to each sequence is optimal. The work in [105] analysed the minimal redundancy of binary codes for the same channel under substitution errors. It also presented an optimal code construction for a single substitution up to constants. The work in [108] introduced a new metric for sets of unordered sequences. The work in [64] used a mechanism called anchoring to combat the ordering loss of short sequences. The work of [103] presents a clustering-correcting codes that ensures if the distance between the index fields of two DNA strands is small, then the distance between the their data fields will be large. To realize the random acess in DNA storage, Yazdi et al. [124] attached spacial address sequences called primers for each DNA sequences. In [17], a construction of error-correcting codes for those primers is designed.

The work in [16] introduced a single large block code strategy which uses LDPC code to encode the data into long codewords and then segment them into short sequences which are suitable for DNA synthesized. They read the short sequences by Illumina iSeq technology and concatenate short sequences by index. Our work can follow the strategy, however, focus on designing LDPC code for the nanopore sequencing. The work in [20] introduced an algorithm and a coding scheme, given deletion and edits, how many sequences are needed to perfectly construct the original sequence. Our problem is different, we don't have to get the original sequence perfectly, as another decoding step will follow reconstruction. Reference [14] shows that soft information improves the multiple sequence reconstruction by using 3X lower read cost compared to hard information.

The rest of the paper is organized as follows. Section 3.1 gives the channel model and an overview of conventional LDPC decoder. Section 3.2 proposes a binary LDPC and a dedicated decoder for DNA storage. Simulation results are presented in Section 3.5. Finally, we draw concluding remarks in Section 4.5.

**Notation.** For a matrix $\Sigma$, denote by $\Sigma^t$ its transpose. For a square matrix $\Sigma$, denote by $|\Sigma|$ its determinant. Vectors are represented by bold font letters. However, a vector of length 2 is represented by normal font if it corresponds to one nucleotide symbol (e.g. the

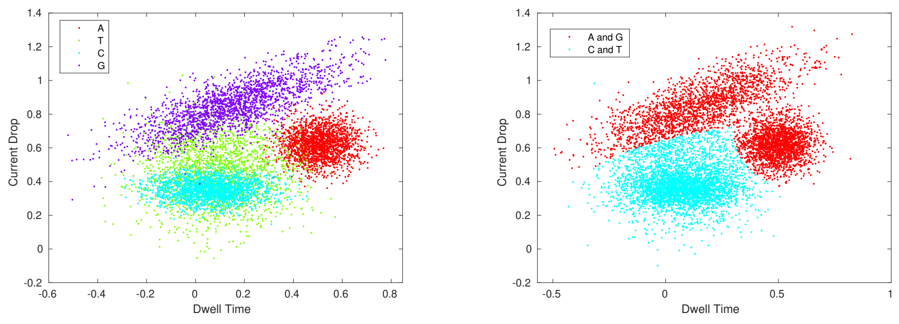


Figure 3.1: Left: Measurement distribution in nanopore sequencing. Right: Area that our methods are used

Table 3.1: Hard error rate and channel capacity from the 2D Gaussian model with Variance 1 and uniform input

| $P_{ac}$ | $P_{at}$ | $P_{ag}$ | $P_{cg}$ | $P_{ct}$ | $P_{tg}$ | Cap |
|---|---|---|---|---|---|---|
| 0.01285 | 0.0583 | 0.0181 | 0.0124 | 0.2828 | 0.0907 | 0.4162 |

binary representation of a nucleotide symbol, or the current and dwell time of one nucleotide symbol). For two sets $I, J$, denote by $I - J = \{i : i \in I, i \notin J\}$ the set difference. If not specified, log represents the natural logarithm ln.

## 3.1 Asymmetric Channel Models

We first introduce the asymmetric channel models of nanopore sequencing. Nanopore sequencing can read the nucleotide symbol in an DNA strand by monitoring small changes in the ionic current flowing through the pore and dwell time in the nanopore [33], and the sampled measurements are shown in Fig. 3.1. The channel takes quaternary input $X \in \{A, T, G, C\}$, and similar to [36] is assumed to be memoryless following the same channel transition probabilities for each input symbol. We consider the i.u.d. inputs where the input symbols are independent and uniformly distributed. Depending on the accuracy of the decision on the output, we propose three channel models.

**2D Gaussian model.** We first consider that the channel output $Y \in \mathbb{R}^2$ corresponds to both current and dwell time data. The distribution of $Y$ conditioned on a channel input symbol is modeled to be 2D Gaussian following the experiment data from [33]. In particular, given the channel input symbol $x \in \{A, T, G, C\}$, the output $Y$ a two dimensional Gaussian random vector. Let $\mu_x \in \mathbb{R}^2, \Sigma_X \in \mathbb{R}^{2\times 2}$ be the corresponding mean vector and the covariance matrix, respectively. Moreover, we use superscripts 1 and 2 to denote the time and the current component of the output. For example, given the channel input $x = A$, the output is $Y_A = (Y_A^1, Y_A^2)^t$, with mean $\mu_A = E[Y_A] = (\mu_A^1, \mu_A^2)^t$, and covariance matrix

$$\Sigma_A = E[(Y_A - \mu_A)(Y_A - \mu_A)^t] = \begin{pmatrix} \sigma_A^{11} & \sigma_A^{12} \\ \sigma_A^{12} & \sigma_A^{22} \end{pmatrix}, \tag{3.1}$$

where $\sigma_A^{11}, \sigma_A^{22}, \sigma_A^{12}$ represent the variance of the time and the current component and their covariance, respectively. The probability density function of $Y$ conditioned on the input $x$ therefore is

$$f(y|x) = \frac{1}{2\pi\sqrt{|\Sigma_x|}} e^{-\frac{1}{2}(y-\mu_x)^t \Sigma_x^{-1}(y_x - \mu_x)}, \tag{3.2}$$

for $x \in \{A, T, G, C\}$, $y \in \mathbb{R}^2$.

The corresponding channel capacity under i.u.d. input is calculated by

$$Cap = I(X;Y) \tag{3.3}$$

$$= h(Y) - h(Y|X) \tag{3.4}$$

$$= -\iint \sum_{x\in\{A,T,G,C\}} \frac{f(y|x)}{4} \log_2 \left( \sum_{x\in\{A,T,G,C\}} \frac{f(y|x)}{4} \right) dy - \sum_{x\in\{A,T,G,C\}} \frac{1}{8} \log_2((2\pi e)^2 |\Sigma_x|). \tag{3.5}$$

The last step follows from the definition of differential entropy and the formula for mutli-

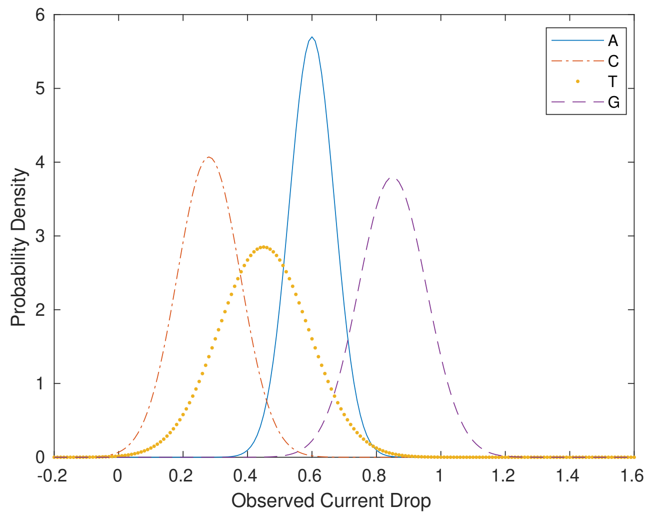


Figure 3.2: 1-D Guassian channel of current drop of nannopore sequencing. The mean and variance data is from [33].

dimensional Gaussian differential entropy, where $e$ is the natural number. The parameters and the capacity of the 2D Gaussian model can be found in Section 3.5.

**1D Gaussian Model.** In some cases, if the only the ionic current value is available from the nanopore sequencing, we can only build a 1-D Gaussian channel, the output is $Y \in \mathbb{R}$. For each input symbol $x \in \{A, T, G, C\}$, the mean of $Y$ corresponds to the first componont of $\mu_x$ in the 2D model, and the variacne corresponds to $var_x^1$ in the 2D model. The capacity can be calculated similarly as the 2D case.

Fig 3.2 shows the 1D Gaussian channel of current drop.

**Hard decoding model.** In some cases, if soft information is not necessary or unavailable, we can employ hard coding. For hard coding, the maximum likelihood estimation decision of each input symbol is made from the current and the time measurements, resulting in substitution errors. Therefore, the channel output $Y \in \{A, T, G, C\}$. The error probabilities between the 4 symbols $A, T, C, G$ are shown in Fig. 3.3. To simplify the model, we assume that the channel transition probability $P(Y = y|X = x) = P(Y = x|X = y)$, for all $x, y \in \{A, T, G, C\}$, which is denoted by $p_{xy}$ or $p_{yx}$. The parameters and the corresponding

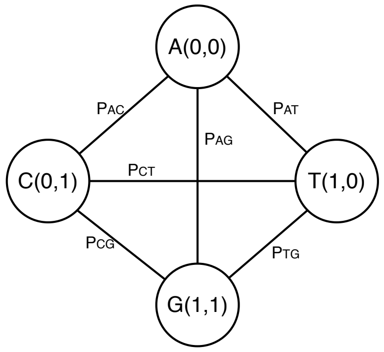


Figure 3.3: Error probabilities in the hard decoding nanopore channel.

channel capacity under i.u.d. channel input is shown in Table 3.1.

**Remark.** In order to approach channel capacity for the general case, we need to optimize the mutual information of the channel input and output, among the possible distributions for the transmitted symbols. However, for LDPC code, we must have uniform input. We now demonstrate that the i.u.d. input is near optimal under hard decoding model. For this purpose, we consider the error probabilities as in Fig 3.3. Denote by $Cap$ the channel capacity, and $X, Y$ the random variables of the channel input and output. Let $p(x)$ denote the probability distribution for the channel input $X$. The capacity

$$Cap = \max_{p(x)} I(X;Y) \tag{3.6}$$

$$= \max_{p(x)} H(Y) - H(Y|X) \tag{3.7}$$

is maximized as 0.4030, when the percentage of $X = [A, C, T, G]$ is [37,19,19,25]. The mutual information of the uniform distribution is 0.3876. We can see those probabilities are closed. Thus one can use the uniform distribution for the transmitted symbols, hence a linear block code with uniform information symbolsis suitable for such channels.

We map the four types of nucleotide to two bits according. When a quaternary code is used, these two bits correspond to the vector representation of elements in the finite field $GF(4)$; when a binary code is used, they are viewed as a 2-bit vector in $\{0,1\}^2$. In this paper, the mapping is chosen to be

$$A \mapsto (0,0), G \mapsto (1,1), T \mapsto (1,0), C \mapsto (0,1). \tag{3.8}$$

According to the channel model in Fig. 3.3, there are total 4! mappings can be done. In section 3.5 we compare the BERs between different mappings and find that the mapping with the lowest raw BER does not give a better result under our methods.

The same quaternary to binary mapping will be used under soft decoding.

## 3.2 Binary LDPC codes for DNA Storage

In this section, we present the binary LDPC codes for DNA storage. We introduce the hard decoding model first for simplicity. We propose our decoder that is inspired by Turbo decoder.

### 3.2.1 Review of LDPC code decoding

Next, we briefly review the LDPC code decoding sum-product algorithm (SPA) (see, e.g., [98]). For simplicity, we describe the algorithm for the binary case. Consider a binary LDPC code defined by an $(n-k) \times n$ parity-check matrix $H$. Let its $(i,j)$-th element be $h_{i,j}$. The code can be represented by a Tanner graph, with $n$ variable nodes (VNs) and $n-k$ check nodes (CNs). An edge exists between the $j$-th variable $VN_j$ and the $i$-th check $CN_i$ if and only if $h_{i,j} = 1$. Let $N(i)$ denote the set of neighboring nodes of the node $i$. When the Tanner graph has no cycles, SPA can be proved to be the bit-wise maximum *a posteriori* (MAP) decoder [96].

Let $\mathbf{x}^1 = (x_1^1, x_2^1, \ldots, x_n^1)$, $\mathbf{y}^1 = (y_1^1, y_2^1, \ldots, y_n^1)$ be the transmitted and the received binary words, respectively. The superscript 1 is added to be consistent with the notation in the next subsection for quaternary symbols. In SPA, the log-likelihood ratio (LLR), $\log \frac{p(x_j^1=0|\mathbf{y}^1)}{p(x_j^1=1|\mathbf{y}^1)}$, is passed between neighboring variable nodes and check nodes in iterations. The LLR passed from node $i$ to node $j$ in the algorithm, called extrinsic information, is the aggregated information of the incoming messages to node $i$ from its neighbors except $j$, i.e., $N(i) - \{j\}$.

Let $iter$ be the index of the current iteration. Define the information passed from the variable node $VN_j$ to the check node $CN_i$ by $L_{j\to i}^{iter}$, where $1 \leq j \leq n$ and $i \in N(i)$. Define the information passed from the check node $VN_i$ to the variable node $CN_j$ by $L_{i\to j}^{iter}$, where $1 \leq i \leq n-k$ and $j \in N(i)$. In the 0-th iteration, $L_{j\to i}^0$ is initialized by $L_{j\to i}^0 = Ch_j$, where for the given received $y_j$, $Ch_j$ is the channel information defined by

$$Ch_j = \log \frac{p(y_j^1|x_j^1 = 0)}{p(y_j^1|x_j^1 = 1)}. \tag{3.9}$$

The message from a CN to a VN is given by

$$L_{i\to j}^{iter} = 2\tanh^{-1}\Big(\prod_{j'\in N(i)-\{j\}} \tanh\big(\frac{1}{2}L_{j'\to i}^{iter-1}\big)\Big). \tag{3.10}$$

The message from a VN to a CN is be updated by

$$L_{j\to i}^{iter} = Ch_j + \sum_{i'\in N(j)-\{i\}} L_{i'\to j}^{iter-1}. \tag{3.11}$$

Then the overall soft information is obtained for each $VN_j$, $1 \leq j \leq n$:

$$L_j^{soft,iter} = Ch_j + \sum_{i'\in N(j)} L_{i'\to j}^{iter-1}, \tag{3.12}$$

and the corresponding symbols is decoded as $\hat{x}_j^{1,iter} = 1$ if $L_j^{soft,iter} < 0$, and decoded as 0 otherwise. Let $\hat{\mathbf{x}}^{1.iter} = (\hat{x}_1^{1,iter}, \hat{x}_2^{1,iter}, \ldots, \hat{x}_n^{1,iter})$. The algorithm stops if $\hat{\mathbf{x}}^{1.iter} \cdot H^T = 0$ or the maximum number of iterations is reached.

### 3.2.2 Encoding

In an $(n, k)$ DNA code, $2k$ bits of information are encoded into a DNA codeword $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, $x_j \in \{A, T, G, C\}, 1 \leq j \leq n$. By abuse of notation, $x_j$ also denotes the two-bit binary representation as in (3.8). The construction of the binary codes is straightforward: we take $k$ bits of information, and encode them into a binary LDPC codeword of length $n$. Repeat for a second binary LDPC code. The two LDPC codes are denoted by $\mathcal{C}^1, \mathcal{C}^2$, and have the same rate $k/n$ and distribution. The parity check matrix of $\mathcal{C}^1, \mathcal{C}^2$ are different. The two codewords are denoted by $\mathbf{x^i} = (x_1^i, x_2^i, \ldots, x_n^i)$, for $i = 1, 2$, respectively. Then we store $x_j = (x_j^1, x_j^2)$ as one DNA quaternary symbol, for all $1 \leq j \leq n$. After the nanopore sequencing, we receive the channel output, $y_j$, $1 \leq j \leq n$. The alphabet of $y_j$ can be 2D, 1D real numbers, or quaternary symbols, depending on the channel model.

Below, we use notations similar to the sum-product algorithm described in Section 3.1, except that we add a superscript 1 or 2 to denote the corresponding notation for the first or the second LDPC code. For example, denote by $VN_j^1, VN_j^2$ the $j$-th variable node in the first and the second LDPC codes, respectively; denote by $L_{j\to i}^{1,iter}, L_{j\to i}^{2,iter}$ the information passed from note $j$ to node $i$ in the two codes.

We refer the above encoder as the separate encoder. The decoding algorithms are designed for the separate encoder. However, we demonstrate our decoders under joint encoding in Section 3.5 as well.

### 3.2.3 Proposed Decoding Algorithms

**The baseline decoder.** We propose a baseline decoder that runs two SPAs on the two codewords independently, with some modifications to the channel information $Ch_j^1, Ch_j^2$. Conditioned on the received quaternary $\mathbf{y}$, the bit-wise MAP rule of the first LDPC is:

$$\begin{aligned}
\hat{x}_i^1 &= \arg\max_{x_i^1 \in \{0,1\}} p(x_i^1|\mathbf{y}) && (3.13)\\
&= \arg\max_{x_i^1 \in \{0,1\}} \sum_{\sim x_i^1} p(\mathbf{x}^1|\mathbf{y}) && (3.14)\\
&= \arg\max_{x_i^1 \in \{0,1\}} \sum_{\sim x_i^1} p(\mathbf{y}|\mathbf{x}^1)P(\mathbf{x}^1) && (3.15)\\
&= \arg\max_{x_i^1 \in \{0,1\}} \sum_{\sim x_i^1} (\prod_{j=1}^{n} p(y_j|x_j^1))\mathbb{1}_{\mathbf{x}^1 \in \mathcal{C}^1}, && (3.16)
\end{aligned}$$

where $\sim x_i^1$ means all possible binary vectors $\mathbf{x}^1 \in \{0,1\}^n$ such that $x_i^1$ is fixed, and $\mathbb{1}_{\mathbf{x}^1 \in \mathcal{C}^1}$ is the indicator function that $\mathbf{x}^1$ is a codeword of the second LDPC code $\mathcal{C}^1$. Here (3.15) follows from Bayes' rule and $\mathbf{y}$ being fixed, and (3.16) follows from the memoryless channel and the uniformity of the transmitted codewords. In a conventional binary LDPC, the bit-wise MAP rule has the term $\prod_j p(y_j^1|x_j^1)$ instead of $\prod_j p(y_j|x_i^1)$ in (3.16), corresponding to the channel information. Effectively, the nanopore channel has the binary input $x_i^1$ and the quaternary output $y_i = (y_i^1, y_i^2)$, and the channel information of (3.9) is modified to

$$Ch_j^1 = \log \frac{p(y_j|x_j^1 = 0)}{p(y_j|x_j^1 = 1)}. \tag{3.17}$$

The same analysis can be applied to the second LDPC. Much like conventional LDPC, for cycle-free Tanner graphs, the baseline decoder is a bit-wise MAP decoder, conditioned on $\mathbf{y}$, for each individual LDPC.

In the computation of the channel information (3.17) for the baseline decoder, it is assumed that a priori distribution of $x_j^2$ is uniform. However, we will also propose other improved algorithms which use soft information for $x_j^2$ from the second LDPC code.

Observe that in Fig. 3.3, if the received symbol is $y_j = C = (0,1)$, and the first bit is decoded to be 1, then it is much more likely that the transmitted symbol is $x_j = T = (1,0)$ than the case of $x_j = G = (1,1)$. One can verify that given $y_j = C$ or $T$, the probability of having complementary bits ($x_j^1 = 1 - x_j^2$) is much larger than identical bits ($x_j^1 = x_j^2$). We note that Turbo codes (e.g., [98]) have a similar property. In particular, the Turbo codes can be viewed as two separate convolutional codes, sharing identical but permuted information bits. The two identical information bits pass extrinsic information between each other hence connecting the two convolutional codes. Inspired by Turbo codes, we propose to pass auxiliary information between the two bits when receiving $C$ or $T$.

In our proposed decoders, the two LDPC codes run their sum-product algorithms as in Section 3.1 concurrently, with the modifications as below. If a received symbol $y_j$ is $C$ or $T$, then an auxiliary term is passed from $VN_j^1$ to $VN_j^2$ in each iteration. We note that different from Turbo codes, the auxiliary term does not appear as an addend in the overall soft information of (3.12) due to two reasons: (i) the two bits $y_j^1, y_j^2$ are not deterministically complementary of each other, (ii) the function in (3.10) is nonlinear. Therefore, we cannot eliminate the auxiliary term entirely when information is passed back from $VN_j^2$ to $VN_j^1$, and the algorithm for the auxiliary information exchange is not straightforward. Regardless, we call the auxiliary term *extrinsic information* to highlight the resemblance to Turbo codes. Next, we propose four decoding algorithms.

**Algorithm 1.** Our first decoding algorithm is shown in Algorithm 3. In this algorithm, we pass the extrinsic information from $VN_j^1$ to $VN_j^2$ if the received quaternary symbol $y_j = C$

or $T$, and the extrinsic information of the $j$-th variable node is defined as

$$Le_j^{1\to 2} = \sum_{i' \in N^1(j)} L_{i' \to j}^{1,iter-1}. \tag{3.18}$$

In every iteration, the extrinsic information is passed from the first LDPC to the second LDPC. To generate the message from a variable node $VN_j^2$ to a check node $CN_i^2$ in the second LDPC code, we combine the above extrinsic information, the channel information of (3.17), and the messages from $VN_{i'}^2$, $i' \in N^2(j)$. Specifically, equation (3.11) is modified to

$$L_{j\to i}^{2,iter} = Ch_j^2 + \sum_{i' \in N^2(j) - \{i\}} L_{i' \to j}^{2,iter-1} - \alpha Le_j^{1\to 2}, \tag{3.19}$$

for some constant parameter $0 < \alpha < 1$. Here $Ch_j^2$ is the channel information similar to (3.17). Moreover, to generate the overall soft information of $VN_j^2$, (3.12) is modified to

$$L_j^{2,soft,iter} = Ch_j^2 + \sum_{i' \in N^2(j)} L_{i' \to j}^{2,iter-1} - \alpha Le_j^{1\to 2}. \tag{3.20}$$

Equivalent to (3.19) (3.20), define the new channel information as

$$\overline{Ch}_j^{2,iter} = Ch_j^2 - \alpha Le_j^{1\to 2}, \tag{3.21}$$

and simply replace $Ch_j^2$ by $\overline{Ch}_j^{2,iter}$ in the sum product algorithm of Section 3.1. The extrinsic information from $VN_j^2$ to $VN_j^1$ is defined similarly, and the algorithm of the first LDPC uses the new channel information similar to (3.21).

In expression (3.18) of the extrinsic information, the coefficient $-\alpha$ captures the fact that $x_j^1$ and $x_j^2$ are complementary with a large probability. Note that $Le_j^{1\to 2}$ is the overall soft information at $VN_j^2$ as in (3.12) except the channel information term. The channel information is excluded because it is unchanged during the iterations, and would be amplified over the iterations during the extrinsic information exchange.

**Algorithm 3** Binary LDPC decoder for DNA storage

```
    Initialize:
1:  For $1 \le j \le n$, $i \in N(j)$ initialize $L^{1,0}_{j\to i} = Ch^1_j$ and $L^{2,0}_{j\to i} = Ch^2_j$ by (3.17)
2:  for $iter = 1 : I_{\max}$ do
3:      Stop if $\hat{\mathbf{x}}^{1,iter}(H^1)^T = 0, \hat{\mathbf{x}}^{2,iter}(H^2)^T = 0$ or $iter = I_{\max}$
    Message passing between two codes
4:      for $1 \le j \le n$ do
5:          if $y_j = C = (0,1)$ or $y_j = T = (1,0)$ then
6:              $\overline{Ch}^{1,iter}_j = Ch^1_j - \alpha \sum_{i' \in N(j)} L^{2,iter-1}_{i'\to j}$
7:              $\overline{Ch}^{2,iter}_j = Ch^2_j - \alpha \sum_{i' \in N(j)} L^{1,iter-1}_{i'\to j}$
8:          else
9:              $\overline{Ch}^{1,iter}_j = Ch^1_j$
10:             $\overline{Ch}^{2,iter}_j = Ch^2_j$
11:         end if
12:     end for
    Check node update
13:     for $1 \le i \le n-k, j \in N(i)$ do
14:         $L^1_{i\to j} = 2\tanh^{-1}(\prod_{j' \in N(i)-\{j\}} \tanh(\frac{1}{2}L^1_{j'\to i}))$
15:         $L^2_{i\to j} = 2\tanh^{-1}(\prod_{j' \in N(i)-\{j\}} \tanh(\frac{1}{2}L^2_{j'\to i}))$
16:     end for
    Variable node update
17:     for $1 \le j \le n, i \in N(j)$ do
18:         $L^1_{j\to i} = \overline{Ch}^{1,iter}_j + \sum_{i' \in N(j)-i} L^1_{i'\to j}$
19:         $L^2_{j\to i} = \overline{Ch}^{2,iter}_j + \sum_{i' \in N(j)-i} L^2_{i'\to j}$
20:     end for
21:     for $i = 1 : n$ do
22:         $L^{1,soft,iter}_j = \overline{Ch}^{1,iter}_j + \sum_{i' \in N(j)} L^{1,iter}_{i'\to j}$
23:         $\hat{x}^1_j = \begin{cases} 1, & \text{if } L^{1,soft,iter}_j < 0, \\ 0, & \text{else.} \end{cases}$
24:         Compute $\hat{x}^2_j$ similarly
25:     end for
26: end for
```

**Algorithm 2.** In the second algorithm, we derive an alternative channel information expression. In the following, we assume $y_j = C$, but the derivation for the case $y_j = T$ is similar and hence omitted.

We define $Le_j^{1\to 2}$ the same as (3.18), and we treat it as the LLR of $x_j^1$ given $y_j = C$:

$$\log \frac{p(x_j^1 = 0|y_j = C)}{p(x_j^1 = 1|y_j = C)} = Le_j^{1\to 2}. \tag{3.22}$$

Since $x_j^1 \in \{1, 0\}$, the corresponding probability can be computed from LLR:

$$p(x_j^1|y_j = C) = \frac{e^{\frac{-Le_j^{1\to 2}}{2}}}{1 + e^{-Le_j^{1\to 2}}} e^{\frac{x_j^1 Le_j^{1\to 2}}{2}}. \tag{3.23}$$

Assume $y_j = C$, from (3.17), a new channel information from the extrinsic information $Le_j^{1\to 2}$ in iteration $iter$ is set to be

$$\begin{align} Ch_j^{1\to 2, iter} &= \log \frac{p(y_j = C|x_j^2 = 0)}{p(y_j = C|x_j^2 = 1)} \tag{3.24}\\ &= \log \frac{p(x_j^2 = 0|y_j = C)}{p(x_j^2 = 1|y_j = C)}, \tag{3.25}\end{align}$$

which follows from Bayes' rule and the uniformity of $x_j^2$. Moreover,

$$\begin{align} & p(x_j^2 = 0|y_j = C) \\ = & \sum_{\gamma \in \{0,1\}} p(x_j^2 = 0|x_j^1 = \gamma, y_j = C) p(x_j^1 = \gamma|y_j = C) \tag{3.26}\end{align}$$

When $\gamma = 0$,

$$\begin{align} & p(x_j^2 = 0|x_j^1 = 0, y_j = C) \tag{3.27}\\ = & \frac{p(x_j^2 = 0, x_j^1 = 0, y_j = C)}{p(x_j^1 = 0, y_j = C)} \tag{3.28}\\ = & \frac{p(x_j = A, y_j = C)}{\sum_{\beta \in \{0,1\}} p(x_j^1 = 0, x_j^2 = \beta, y_j = C)} \tag{3.29}\\ = & \frac{\frac{1}{4} p(y_j = C|x_j = A)}{\frac{1}{4} p(y_j = C|x_j = A) + \frac{1}{4} p(y_j = C|x_j = C)} \tag{3.30}\\ = & \frac{p_{AC}}{1 - p_{CG} - p_{CT}} \tag{3.31}\end{align}$$

where (3.30) follows from the uniformity of the the transmitted symbol $x_j$ and (3.31) follows from the channel transition in Figure 3.3. Similarly, we can calculate for the case $\gamma = 1$, and thus obtain

$$p(x_j^2 = 0|y_j = C) = \frac{p_{AC}}{1 - p_{CG} - p_{CT}} p(x_j^1 = 0|y_j = C) + \frac{p_{CT}}{p_{CG} + p_{CT}} p(x_j^1 = 1|y_j = C). \tag{3.32}$$

In the above calculation, the probability $p(x_j^1|y_j = C)$ should be computed based on the extrinsic information from the first LDPC as in (3.23). Similarly, we have

$$p(x_j^2 = 1|y_j = C) = \frac{1 - p_{AC} - p_{CG} - p_{CT}}{1 - p_{CG} - p_{CT}} p(x_j^1 = 0|y_j = C) + \frac{p_{GC}}{p_{CG} + p_{CT}} p(x_j^1 = 1|y_j = C). \tag{3.33}$$

Finally, we use the new channel information

$$\overline{Ch}_j^{2,iter} = Ch_j^{2,iter} + Ch_j^{1\to 2,iter} \tag{3.34}$$

by equations (3.17) (3.25) (3.32) (3.33) (3.23) in our SPA. As a special case, if $p_{AC} = p_{CG} = 0$, then we have $\overline{Ch}_j^{2,iter} = Ch_j^{2,iter} + \log \frac{P(x_j^1=1|y_j=C)}{P(x_j^1=0|y_j=C)} = Ch_j^{2,iter} - Le_j^{1\to 2}$, corresponding to complementary bits of $x_j^1, x_j^2$ and $\alpha = 1$ in algorithm 1.

**Algorithm 3.** In the third decoding algorithm, we add an imaginary variable node $u_j$ and an imaginary check node $e_j$ in the Tanner graph, for every $j$ such that $y_j = C$ or $T$. The imaginary nodes serve the purpose of extrinsic information exchange. As Fig. 3.4 shows, after iteration $iter - 1$, we pass the variable message from $VN_j^1$ and $u_j$ to the check node $e_j$. Then we pass the check message from $e_j$ to $VN_j^2$. The message from $VN_j^1$ to $e_j$ is $Le^{1\to 2}$ as in (3.18). Note that the channel information is not included at this step. The variable node

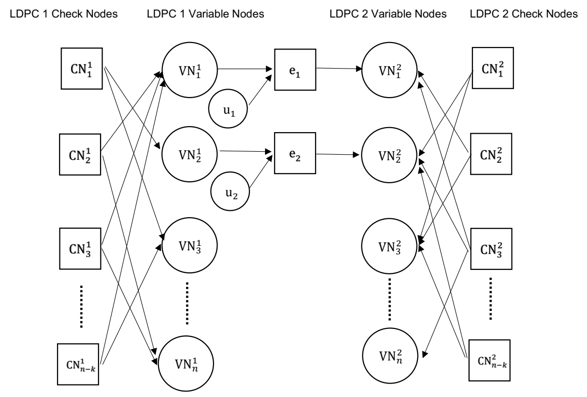


Figure 3.4: Tanner graph for Algorithm 3. In this example, for $j = 1, 2$, the observed symbol $y_j$ is $C$ or $T$, and new nodes $u_j, e_j$ are added.

$u_j$ indicates that $VN_j^1$,$VN_j^2$ are complementary with a large probability. In particular,

$$u_j + x_j^1 + x_j^2 = 0, \tag{3.35}$$

where $u_j$ is set to have a fixed LLR of

$$L_{u_j} = \log \frac{p(u_j = 0|y_j = C)}{p(u_j = 1|y_j = C)} = \frac{p_{AC} + p_{CG}}{1 - p_{AC} - p_{CG}}. \tag{3.36}$$

Here (3.36) is due to (3.35) and the uniformity of $x_j$. One can see that the message from the check node $e_j$ to $VN_j^2$ is a function of the extrinsic information $Le^{1\to 2}$.

**Algorithm 4.** Palanki et al. [92] proposed an algorithm for two separate LDPC codes on a binary-input two-user channel which has a similar idea to our decoder. Inspired by [92], we derive the joint decoding for the nanopore channel model. Algorithm 4 corresponds to the bit-wise MAP decoder when the two LDPC codes are considered jointly and the corresponding factor graph (Tanner graph together with the joint channel node) is cycle-free. Different from previous methods, information is exchanged between the two LDPC codes regardless of the received symbol $y_j$.

The bit-wise MAP decoder is

$$\begin{aligned}
\hat{x}_i^1 &= \arg \max_{x_i^1 \in \{0,1\}} p(x_i^1 | \mathbf{y}) && (3.37)\\
&= \arg \max_{x_i^1 \in \{0,1\}} \sum_{\sim x_i^1} p(\mathbf{x}^1, \mathbf{x}^2 | \mathbf{y}) && (3.38)\\
&= \arg \max_{x_i^1 \in \{0,1\}} \sum_{\sim x_i^1} p(\mathbf{y} | \mathbf{x}^1, \mathbf{x}^2) p(\mathbf{x}^1, \mathbf{x}^2) && (3.39)\\
&= \arg \max_{x_i^1 \in \{0,1\}} \sum_{\sim x_i^1} \left( \prod_{j=1}^{n} p(y_j | x_j^1, x_j^2) \right) \mathbb{1}_{(\mathbf{x}^1, \mathbf{x}^2) \in \mathcal{C}}. && (3.40)
\end{aligned}$$

Here $\mathcal{C}$ represents the overall LDPC code for $(\mathbf{x}^1, \mathbf{x}^2)$. In particular, (3.40) holds regardless whether the bits of $\mathbf{x}^1$ and $\mathbf{x}^2$ are coded separately or jointly. However, for the ease of notation, we assume that separate coding is used. The bit-wise MAp problem reduces to calculating the marginalization with respect to $x_i^1$ as in (3.40). Different from the traditional LDPC code, the channel information $p(y_j | x_j^1, x_j^2)$ is a function of two variable nodes. In the Tanner graph, we connect one joint channel node to both $VN_j^1$ and $VN_j^2$, while in the first 3 algorithms we construct two individual channels nodes for $VN_j^1$ and $VN_j^2$, respectively. Thus, the joint channel node is responsible for exchanging information between the two LDPC codes in Algorithm 4.

Instead of expressing the messages as log likelihood ratios, we view them as simply the likelihood. We will denote the node indices in the subscript, and the first/second LDPC index in the superscript. The iteration index is omitted. The $j$-th channel node is denoted by $ch$ and we omit the index $j$ as it is clear from the context. For example, we denote by $p_{j \to i}^1(x_j^1)$ the message about the probability of $x_j^1$ from $VN_j^1$ to $CN_i^1$.

According to the message-passing rule, the message from $VN_j^1$ to the $j$-th channel node is

$$p_{j \to ch}^1(x_j^1) = \prod_{i' \in N^1(j)} p_{i' \to j}^1(x_j^1). \qquad (3.41)$$

After that, the message from the $j$-th channel node to $VN_j^2$ is

$$p_{ch}^2(x_j^2) \triangleq p_{ch\to j}^2(x_j^2) = \sum_{x_j^1=0}^{1} \left(p(y_j|x_j^1, x_j^2) p_{j\to ch}^1(x_j^1)\right) \tag{3.42}$$

$$= \sum_{x_j^1=0}^{1} \left( p(y_j|x_j^1, x_j^2) \prod_{i'\in N^1(j)} p_{i'\to j}^1(x_j^1) \right). \tag{3.43}$$

Thus we have obtained the channel information in Algorithm 4. Representing it using LLR,

$$\overline{Ch}_j^{2,iter} \tag{3.44}$$

$$= \log \frac{p(y_j|x_j^1=0, x_j^2=0)\prod_{i'\in N^1(j)} p_{i'\to j}^1(x_j^1=0) + p(y_j|x_j^1=1, x_j^2=0)\prod_{i'\in N^1(j)} p_{i'\to j}^1(x_j^1=1)}{p(y_j|x_j^1=0, x_j^2=1)\prod_{i'\in N^1(j)} p_{i'\to j}^1(x_j^1=0) + p(y_j|x_j^1=1, x_j^2=1)\prod_{i'\in N^1(j)} p_{i'\to j}^1(x_j^1=1)} \tag{3.45}$$

$$= max^*(\log p(y_j|x_j^1=0, x_j^2=0) + \sum_{i'\in N^1(j)} L_{i'\to j}^{1,iter-1}, \log p(y_j|x_j^1=1, x_j^2=0))$$

$$- max^*(\log p(y_j|x_j^1=0, x_j^2=1) + \sum_{i'\in N^1(j)} L_{i'\to j}^{1,iter-1}, \log p(y_j|x_j^1=1, x_j^2=1)) \tag{3.46}$$

where $max^*(a,b) \triangleq \log(e^a + e^b)$. Note that $\sum_{i'\in N^1(j)} L_{i'\to j}^{1,iter-1} = Le_j^{1\to 2}$ which is the extrinsic information defined in (3.18). As can be seen, Algorithm 4 utilizes the extrinsic information in a non-linear fashion.

### 3.2.4 1-Dimensional Gaussian

Next, we briefly explain how to apply the proposed algorithms to channels with soft information. First, we consider the 1D Gaussian channel as in Fig 3.2. Different from hard decoding channel model, we can directly recognize the received symbol $y_j$ as $C$ or $T$. Instead, $y_j$ is a continuous random variable. We use the maximum likelihood hard decision to decide whether $y_j$ corresponds to $C, T$ and, correspondingly, whether the extrinsic information is exchanged. For example, we set a threshold $y_{th}$ which is the current drop such that the channel transition probabilities satisfy $f(y_{th}|A) = f(y_{th}|T)$. Here $f(y|x)$ represents the Gaussian density function of the channel output given that $x$ is the channel input. The intersection of the probability density of $A$ and that of $T$ to help deciding whether to send the extrinsic information or not.

In the sum-product algorithms, we can simply replace all the channel transition probabilities by the Gaussian density function. For example, the channel information in (3.17) is replaced by

$$Ch_j^1 = \log \frac{f(y_j|A) + f(y_j|C)}{f(y_j|G) + f(y_j|T)} \tag{3.47}$$

Algorithms 1 and 4 are identical to the hard decoding model after the channel information modification (3.47). Next we only point out the modified steps for Algorithms 2 and 3.

**Algorithm 2.** Equation (3.32) is changed to

$$p(x_j^2 = 0|y_j) = \frac{f(y|A)}{f(y|A) + f(y|C)} p(x_j^1 = 0|y_j) + \frac{f(y|T)}{f(y|G) + f(y|T)} p(x_j^1 = 1|y_j). \tag{3.48}$$

where $p(x_j^1|y_j)$ is again computed using the extrinsic information from the first LDPC code by (3.23).

**Algorithm 3.** We modify Equation (3.36), the LLR of the variable node $u_j$, as below,

$$L_{u_j} = \log \frac{f(y_j|A) + f(y_j|G)}{f(y_j|C) + f(y_j|T)}. \tag{3.49}$$

### 3.2.5 2-Dimensional Gaussian

when the channel output is 2D Gaussian, the decoding algorithms are exactly the same as the 1D Gaussian case, except that the density fucntion $f(y|x)$ corresponds to the 2D vector $y = (y^1, y^2)$ for the dwell time and the current drop.

In the algorithms, we first make maximum likelihood hard decision given $y_j$. If $y_j$ corresponds to $C, T$, then extrinsic information is exchanged. For example, if $f(y|A) > f(y|C)$, the symbol is more likely to be $A$ than $C$. Using the Gaussian density expression, the boundary between $A, C$ is

$$\frac{1}{\sqrt{|\Sigma_A|}} e^{-\frac{1}{2}(y-\mu_A)\Sigma^{-1}(y-\mu_A)^t} = \frac{1}{\sqrt{|\Sigma_C|}} e^{-\frac{1}{2}(y-\mu_C)\Sigma_C^{-1}(y-\mu_C)^t}. \tag{3.50}$$

$$log(|\Sigma_A|) + (y-\mu_A)\Sigma^{-1}(y-\mu_A)^t = log(|\Sigma_C|) + (y-\mu_C)\Sigma^{-1}(y-\mu_C)^t. \tag{3.51}$$

Simplifying the expression, we get

$$P_1(y^1)^2 + P_2(y^2)^2 + P_3(y^1y^2) + P_4y^1 + P_5y^1 + P_6 = 0, \tag{3.52}$$

where the constant coefficients are

$$P_1 = \sigma_A^{11} - \sigma_C^{11}, \tag{3.53}$$

$$P_2 = \sigma_A^{22} - \sigma_C^{22}, \tag{3.54}$$

$$P_3 = 2\sigma_A^{12} - 2\sigma_C^{12}, \tag{3.55}$$

$$P_4 = -2\mu_A^2\sigma_A^{12} - 2\mu_A^1\sigma_A^{11} + 2\mu_C^2\sigma_C^{12} + 2\mu_C^1\sigma_C^{11}, \tag{3.56}$$

$$P_5 = -2\mu_A^1\sigma_A^{12} - 2\mu_A^2\sigma_A^{22} + 2\mu_C^1\sigma_C^{12} + 2\mu_C^2\sigma_C^{22}, \tag{3.57}$$

$$P_6 = (\mu_A^1)^2\sigma_A^{11} + 2\sigma_A^{12}\mu_A^1\mu_A^2 + (\mu_A^2)^2\sigma_A^{22} + log(|\Sigma_A|) \tag{3.58}$$

$$- (\mu_C^1)^2\sigma_C^{11} - 2\sigma_C^{12}\mu_C^1\mu_C^2 - (\mu_C^2)^2\sigma_C^{22} - log(|\Sigma_C|). \tag{3.59}$$

## 3.3 Multiple Reads

As shown in Section 3.5, the code rate is barely higher than 0.5 when we use the channel model according to [33]. Fortunately, DNA storage can provide multiple reads of the same sequence relatively easily. Namely, one can duplicate the sequence, and then apply the sequencer multiple times. As a result, the coding rate is expected to be improved. For example, the work in [16] used counts for 0's and 1's at a particular nucleotide position from multiple reads to obtain the channel information. On the contrary, we obtain the soft information from each of the reads, and calculate the overall log likelihood ratio for the channel information. For comparison, we also explain a simple method using the mean of the soft measurements from the reads. In this section, it is assumed that for each read the dwelling time and the current drop are measured.

### 3.3.1 Multivariate Gaussian Channel

Let $f(y|x)$ be the 2D Gaussian density function Consider $M$ independent reads for the sequence. Let $y \triangleq (y^{1,1}, y^{1,2}, y^{2,1}, y^{2,2}, \ldots, y^{M,1}, y^{M,2})$ be the channel output from $M$ reads for the $j$-th corrodinate of the DNA sequence, $1 \leq j \leq n$. The index $j$ is omitted in the following discussion for the ease of notations. Assume that $y^i = (y^{i,1}, y^{i,2}), i = 1, 2, \ldots, M$, are independent and identically distributed according to the 2D Gaussian density function $f(y^{i,1}, y^{i,2}|x)$, given that $x$ is the correct nucleotide symbol. The parameters of the 2D Gaussian distribution are from section 3.5. Denote the overall multivariate Gaussian distribution as

$$f(y|x) = \prod_{i=1}^{M} f(y^{i,1}, y^{i,2}|x). \tag{3.60}$$

Note that the notation $f$ is abused but its meaning is clear from the arguments.

The channel capacity will be calculated by

$$
\begin{aligned}
Cap &= I(X;Y) && (3.61)\\
&= H(Y) - H(Y|X) && (3.62)\\
&= -\underbrace{\int \cdots \int}_{2M} \sum_{x\in\{A,C,G,T\}} \frac{f(y|X)}{4} \log_2 \left( \sum_{x\in\{A,C,G,T\}} \frac{f(y|x)}{4} \right) dy && (3.63)\\
&- \sum_{x\in\{A,C,G,T\}} \frac{1}{8} \log_2 \left( (2\pi e)^{(2M)} |\Sigma_x| \right) && (3.64)
\end{aligned}
$$

In order to run the proposed decoding algorithms, we replace the channel transition probabilities in Section 3.2.3 by the multivariate density function. For example, similar to (3.47), the channel information for the first LDPC channel will be calculated by

$$
\begin{aligned}
Ch^1 &= \log \frac{f(y|A) + f(y|C)}{f(y|G) + f(y|T)} && (3.65)\\
&= \log \frac{\prod_{i=1}^{M} f(y^{i,1}, y^{i,2}|A) + \prod_{i=1}^{M} f(y^{i,1}, y^{i,2}|C)}{\prod_{i=1}^{M} f(y^{i,1}, y^{i,2}|G) + \prod_{i=1}^{M} f(y^{i,1}, y^{i,2}|T)} && (3.66)\\
& && (3.67)
\end{aligned}
$$

Let

$$
\begin{aligned}
m_A^i &= \ln f(y^{i,1}, y^{i,2}|A) && (3.68)\\
&= \ln c + h_A(y^{i,1}, y^{i,2}), && (3.69)
\end{aligned}
$$

where $c = \frac{1}{\sqrt{|\Sigma_A|(2\pi)^2}}$, $h_A = -\frac{1}{2}(y^i - \mu_A)\Sigma_A^{-1}(y^i - \mu_A)^t$. Similarly, $m_T^i, m_C^i, m_G^i$ can be defined.

Then we can have

$$Ch^1 = max^* \left( \sum_{i=1}^{M} m_A^i, \sum_{i=1}^{M} m_C^i \right) - max^* \left( \sum_{i=1}^{M} m_T^i, \sum_{i=1}^{M} m_G^i \right), \tag{3.70}$$

where $max^*$ is define after (3.46). Similarly, we have

$$Ch^2 = max^* \left( \sum_{i=1}^{M} m_A^i, \sum_{i=1}^{M} m_T^i \right) - max^* \left( \sum_{i=1}^{M} m_C^i, \sum_{i=1}^{M} m_G^i \right). \tag{3.71}$$

### 3.3.2 Simple Mean

Alternatively, we use the mean of $M$ reads,

$$y^1 = \frac{1}{M} \sum_{i}^{M} y^{i,1}, \tag{3.72}$$

$$y^2 = \frac{1}{M} \sum_{i}^{M} y^{i,2}. \tag{3.73}$$

In this case, the distribution given $x \in \{A, T, G, C\}$ is stored can be easily verified to be $(y^1, y^2) \sim \mathcal{N}(\mu_x, \frac{1}{M}\Sigma_x)$. Here $\mu_x = (\mu_x^1, \mu_x^2)$ is the original mean of dwelling time and current from the 2D Gaussian model. And $\Sigma_x$ is the original covariance matrix.

The decoding algorithms can be run according to the above channel transition distribution.

## 3.4 Density Evolution

In order to quantify the effectiveness of the proposed algorithms in the asymptotic regime for large block length, we investigate density evolution for the asymmetric nanopore channel and for our Turbo-like algorithms in this section. We first briefly review the density evolution algorithm (see, e.g.[98]) which can compute the asymptotic average performance of LDPC codes. The algorithm models the messages passed during decoding as random variables. It tracks the evolution of the probability density function of the messages in the iterative decoder.

If channel symmetry conditions (e.g., binary symmetric channel) are satisfied, then the density evolution algorithm assumes that the all-zeros codeword $c = [0\,0 \dots 0]$ is sent. Therefore, no error will be made if

$$\lim_{l \to \infty} \int_{-\infty}^{0} f_v^{iter}(\tau) d\tau = 0 \tag{3.74}$$

where the $f_v^{iter}$ denotes the probability density function of a message $\xi_v$ to be passed from variable node $v$ to some check node in the $iter$-th iteration. The $f_v^{iter}$ depends on the channel parameter $\alpha$, for instance, crossover probability in BSC and variance factor $var$ of the Gaussian model in our model. Then decoding threshold $\alpha^*$ can be calculated by

$$\alpha^* = \sup\{\alpha : \lim_{l \to \infty} \int_{-\infty}^{0} f_v^{iter}(\tau) d\tau = 0\} \tag{3.75}$$

Then the $f_v^{iter}$ is calculated by

$$f_v^{iter} = f_{ch} * \sum_{i=1}^{d_v} \lambda_i (f_c^{iter-1})^{*(i-1)} \tag{3.76}$$

where the $d_v$ is the maximum VN degree. $f_{ch}$ denotes the pdfs for the channel message which is initialed to be zero. $f_c$ denotes $d_v - 1$ messages from neighboring CNs. Note that, those $d_v$ messages are the same because the algorithm assumes that all-zeros codeword is sent.

The $f_c^{iter}$ can then be updated by $f_v^{iter}$ corresponding to the equation 3.10 in LDPC decoding algorithm:

$$f_c^{iter} = \Phi(f_v^{iter}) \tag{3.77}$$

where the $\Phi$ is the computation of the pdfs $f_c^{iter}$ for a generic message $\xi_c$ from a check node. The detail of calculation of $\Phi(x)$ has been introduced in [98] page 391. We can then update $f_v^{iter}$ and check if equation 3.74 is satisfied.

Next we introduce the density evolution for our decoders.

Let $f_{ch}^{iter}$, denote the channel information probability density distribution(pdf) for the correct message, $f_{vc}^{iter}$ denote the pdf for the message from variable node to check node, $f_{cv}^{iter}$ denote the pdf for the message from check node to variable node and $f_E^{iter}$ denote the pdf extrinsic message from the first bit to the second bit.

### 3.4.1 Input information Assumption

Density evolution assumes the all-zeros codeword is sent due to the symmetry condition. Our method, although satisfies the symmetry condition, can not make the identical input assumption because our algorithms only pass the extrinsic information under a specific condition(for instance, $C$ or $T$ in hard decoding). For instance, if we assume the input codeword $c = [A, A, ...A]$, our method will not work. As a result, We have to average the message for the four symbols as shown in (3.78). We add $\omega$ to transfer all the correct message to the positive part of the pdf to satisfy the (3.78). Similar ideas have been applied to interference channels in [58].

Note that, because the codeword and the received hard decision symbols are i.u.d. The message distribution is defined over all the uniform codeword bits. For example, define the random variable $Ch^i, i = 1, 2$ as the channel message for the $i$-th LDPC code. Here we compute the distribution of $Ch^i$ among all i.u.d. codewords. In order to obtain a unified stopping criteria as in (3.74), we follow the method of [58] and map $Ch^i$ to $-Ch^i$ if the corresponding codeword bit $x^i = 1$. For simplicity, we drop the superscript and use $Ch$ to represent the message of the first bit. The derivation for the second bit is similar. Let $\ell \in \mathbb{R}$ denote a realization of $Ch$.

$$f_{Ch} = \sum_{x=A,C,T,G} \frac{1}{4} f_{ch}(\omega\xi|x) \tag{3.78}$$

$$\omega = \begin{cases} 1, & \text{if } x^1 = 0, \\ -1, & \text{if } x^1 = 1. \end{cases} \tag{3.79}$$

Since the message is now for the "correct message" $\omega Ch$, the steps for check node to variable node and variable node to check node are the same as if all $x^1 = 0$.

Note that, the average method we use is node perspective but not edge perspective. In note perspective, the notation $\bar{\lambda}_i$ and $\bar{\rho}_i$ refers to the fraction of all variable nodes and check nodes that have degree $i$. The polynomials are denoted by $\bar{\lambda}(x) = \sum_{i=1}^{L_{\max}} \bar{\lambda}_i x^i$ and $\bar{\rho}(x) = \sum_{i=1}^{R_{\max}} \bar{\rho}_i x^i$, respectively. Similar to equation 3.76, the pdf of the message $\xi_v$ is calculated by

$$f_v^{iter} = f_{ch} * \sum_{i=1}^{d_v} \bar{\lambda}_i (f_c^{iter-1})^{*(i-1)} \tag{3.80}$$

### 3.4.2 Algorithm 1

In algorithm 1, we add an coefficient $-\alpha$ to captures the fact that the two bits $x^1$ and $x^2$ are complementary with a large probability when $x = C$ or $T$. Similarly, the extrinsic information from $\xi_E$ is denoted by .

$$\xi_E = i \begin{cases} -\alpha\xi_v, & \text{if } y = \text{T or C} \\ 0, & \text{otherwise} \end{cases} \tag{3.81}$$

Let $f_\alpha^{iter}$ be the pdf of $\xi_E$. Correspondingly, we have

$$f_\alpha^{iter}(\xi_v) = \frac{1}{\alpha} f_v^{iter}(-\frac{1}{\alpha}\xi_v) \tag{3.82}$$

The pdf for the extrinsic information will be caculated by

$$f_E^{iter} = \sum_{i=1}^{d_v} \bar{\lambda}_i (\frac{1}{2} f_\alpha^{iter} + \frac{1}{2}\sigma(\xi))^{*(i-1)} \tag{3.83}$$

$$\sigma(\xi) = \begin{cases} 1, & \text{if } \xi == 1 \\ 0, & \text{otherwise} \end{cases} \tag{3.84}$$

Similar to (3.19) (3.20), calculate the $f_v^{2,iter+1}$ for channel 2

$$f_v^{2,iter} = f_E^{1,iter} * f_v^{2,iter} \tag{3.85}$$

The rest calculation for updating $f_c^{iter}$ will follow the traditional density evolution algorithm we showed in Algorithm 4.

**Algorithm 4** Binary LDPC Density Evolution for DNA storage

```
1: Set the variance factor $var$ to some value expected to be less than threshold $var^*$. Set the iteration counter $iter = 0$. Initialize the channel information pdf $f_{ch}^1$, $f_{ch}^2$ for both LDPC codes by (3.78).
2: Given $f_{ch}^1$, $f_{ch}^2$, obtain $f_v^{1,iter}, f_v^{2,iter}$ via (3.80) with $f_c^0 = 0$ because the initial message is 0..
3: Given message $\xi_v$, obtain $f_E^{1,iter}, f_E^{2,iter}$ via (3.82) (3.83)
4: Given $f_E^{1,iter}, f_E^{2,iter}$, update $f_v^{1,iter}, f_v^{2,iter}$ via (3.85)
5: Increase $iter$ by 1. Given $f_v^{1,iter-1}, f_v^{2,iter-1}$, obtain $f_c^{1,iter}, f_c^{2,iter}$ by (3.77).
6: Given $f_c^{1,iter}, f_c^{2,iter}$, obtain $f_v^{1,iter}, f_v^{2,iter}$ via (3.80).
7: If
   $$\int_{-\infty}^{0} f_v^{iter}(\tau)d\tau \leq f_e \text{ and } \int_{-\infty}^{0} f_v^{iter}(\tau)d\tau \leq f_e \tag{3.86}$$
   for some predefined error probability $f_e$(e.g. $f_e = 10^{-6}$) and $iter < iter_{max}$. increase the channel parameter $\beta$ by some small value and go to 2.
8: If (3.86) does not hold but $iter < iter_{max}$ then go back to 3.
9: If (3.86) does not hold and $iter >= iter_{max}$ then previous $\beta$ is the threshold $\beta^*$.
```

### 3.4.3 Algorithm 2

In Algorithm 2, similar to (3.23), we have

$$\xi(x^1|y_j = C) = \frac{e^{\frac{-\xi_v^{1\to 2}}{2}}}{1 + e^{-\xi_v^{1\to 2}}} e^{\frac{x_j^1 \xi_v^{1\to 2}}{2}}. \tag{3.87}$$

.

Use the similar ideas from (3.17) (3.25) (3.32) (3.33) (3.23), we have

$$\xi_E = \log \frac{p(x_j^2 = 0|y_j = C)}{p(x_j^2 = 1|y_j = C)}, \tag{3.88}$$

where the

$$p(x_j^2 = 0|y_j = C) = \frac{P_{AC}}{1 - P_{CG} - P_{CT}}\xi(x^1 = 0|y_j = C) + \frac{P_{CT}}{P_{CG} + P_{CT}}\xi(x_j^1 = 1|y_j = C). \tag{3.89}$$

$$p(x_j^2 = 1|y_j = C) = \frac{1 - P_{AC} - P_{CG} - P_{CT}}{1 - P_{AC} - P_{CT}}\xi(x_j^1 = 0|y_j = C) + \frac{P_{AC}}{P_{AC} + P_{CT}}\xi(x_j^1 = 1|y_j = C).$$

### 3.4.4 Algorithm 3

In Algorithm 3, we change (3.81) to

$$\xi_E = 2tanh^{-1}(tanh(\frac{1}{2}\text{Lu})tanh(\frac{1}{2}\xi_v)) \tag{3.90}$$

### 3.4.5 Joint Code

In joint code, (3.81) is changed to

$$\begin{aligned}\xi_E &= max^*(\log p_{ch}(x^1 = 0, x^2 = 0) + \xi_v, \log p_{ch}(x^1 = 0, x^2 = 1)) \\ &\quad - max^*(\log p_{ch}(x^1 = 1, x^2 = 0) + \xi_v, \log p_{ch}(x^1 = 1, x^2 = 1))\end{aligned} \tag{3.91}$$

We can then calculated the pdf of $\xi_E$ correspondingly for those different methods. Since the calculations are similar and hence omitted.

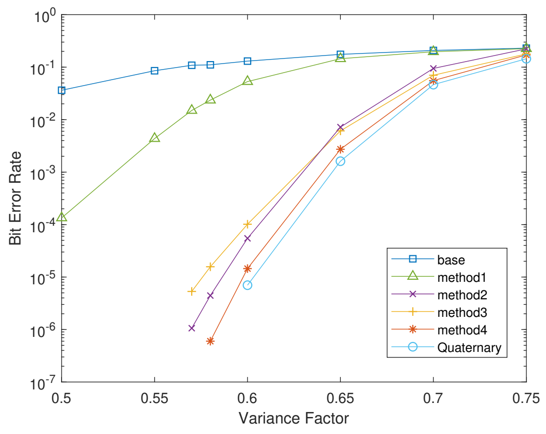


Figure 3.5: Bit error rate comparison between our three algorithms with hard decoding, quaternary LDPC and the baseline algorithm.

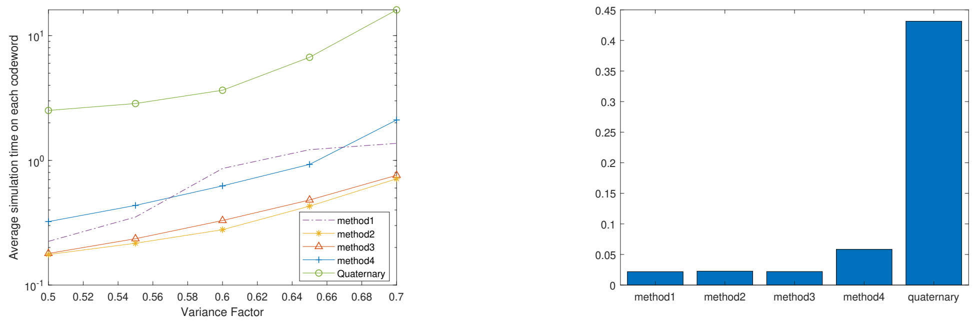
## 3.5 Simulation Results

### 3.5.1 Quaternary Code

We first compare our three methods with the quaternary code. We construct the LDPC code with the commonly used $(3, 6)$ distribution, Namely, every variable node has degree 6, and every check node has degree 3. To construct the quaternary code, we first choose the same degree distribution as the binary code. Then in the parity check matrix the ones are randomly changed to one of the three non-zero elements of $GF(4)$.

The bit error rate of quaternary codes, the 3 proposed decoding algorithms, and the baseline binary decoder is shown in Fig. 3.7c. The code has code $n = 2000$ bits code length and $k = 1000$ bits data length. For Algorithm 1, we pick the best parameter $\alpha = 0.1$. We can see that the error rate of Algorithms 2,3,4 are better than the baseline binary decoder and comparable to the quaternary codes. The Algorithm 4 performs best because the two LDPC codes are considered jointly.

Moreover, the simulation time of the algorithms are shown in 3.6a and 3.6b. Algorithms 1,2,3 are 20 times faster on each iteration and 14 times faster on each codeword than quaternary



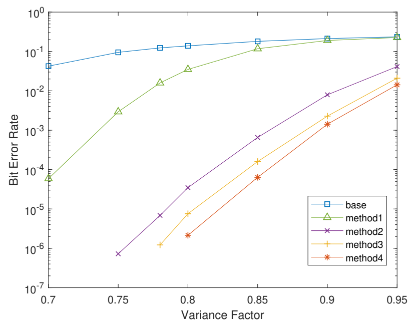
(a) Average decoding time per codeword (b) Average decoding time per iteration

Figure 3.6: Average running time comparison between our methods and quaternary LDPC.

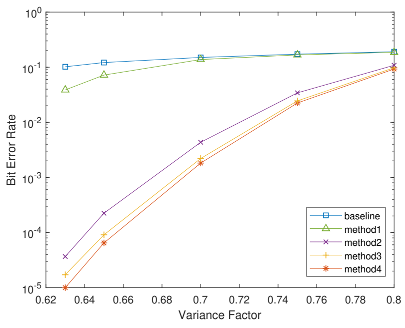
codes. Given the demand of high-volume data storage, the speed-up can improve system performance significantly. Within our methods, algorithm 4 performs slowest due to the marginalized calculation. The iteration round may affect the running time on each codeword. For example, in 3.6a, when variacne factor is bewteen 0.6 and 0.65, the simulation time of method 1 is larger than method 4 due to the higher bit error rate.

### 3.5.2 Soft Decoding

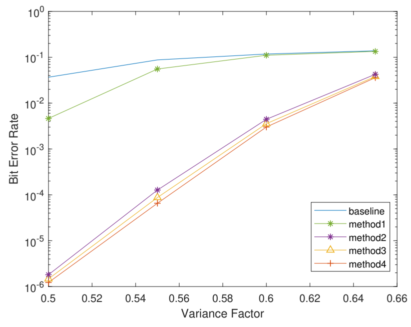
The soft channel parameter we used is based on the from [33] : $\mu_A = (0.5, 0.62)^t$, $\mu_C = (0.06, 0.31)^t$, $\mu_T = (0.09, 0.49)^t$, $\mu_G = (0.15, 0.83)^t$, $\Sigma_A = \begin{pmatrix} 0.01 & 0 \\ 0 & 0.01 \end{pmatrix}$, $\Sigma_C = \begin{pmatrix} 0.0225 & 0 \\ 0 & 0.0196 \end{pmatrix}$, $\Sigma_T = \begin{pmatrix} 0.04 & 0 \\ 0 & 0.04 \end{pmatrix}$, $\Sigma_G = \begin{pmatrix} 0.0194 & 0.0054 \\ 0.0054 & 0.0131 \end{pmatrix}$. The channel capacity is 1.2454 calculated from (3.5). The simulation results are shown in Fig. 3.7. We use (3,6),(3,8),(3,10) codes to see how our methods perform under different code rate. From the results we can see that the comparison between different methods in soft decoding is similar to hard decoding. All methods perform better BER in soft decoding than hard coding, which shows that working with the raw nanopore signal is beneficial to decoding.



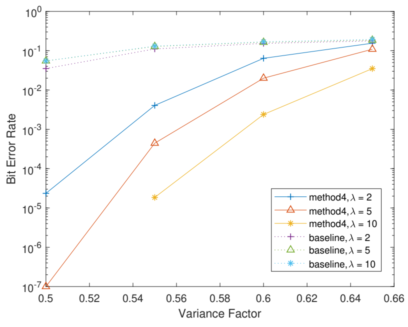
(a) (3,6) code, soft decoding, 2D Gaussian Model.



(b) (3,8) code, soft decoding, 2D Gaussian Model



(c) (3,10) code, soft decoding, 2D Gaussian Model



(d) (3,6) code, soft Decoding, General Model shown in Fig. 3.8.

Figure 3.7: Bit error rate comparison between our methods and baseline algorithm. Code parameters $n = 2000, k = 1000$.

Table 3.2: Bit error rate comparison between Multivariate distribution and simple mean with different numbers of reads $M$. The code rate is 0.8754, variance factor is 1. Raw data length is 2000 bits.

| M | 2 | 5 | 8 | 10 | 100 |
|---|---|---|---|---|---|
| Simple Mean | 0.253 | 0.208 | 0.195 | 0.185 | 0.125 |
| Multivariate Distribution | 0.220 | 0.0799 | 6.12e-4 | 0 | 0 |

### 3.5.3 Multiple Reads

We simulate the decoding algorithms by different multiple reads $M = 2, 5, 8, 10, 100$. We use irregular LDPC code with the distribution in table 3.4 with the code rate 0.8754. The variance factor of the 2D Gaussian channel is 1, raw data length is 2000 bits. The result is shown in table 3.2. We can obviously see that Simple Mean does not work efficiently compared to the Multivariate Distribution. The Multivariate Distribution significantly improve the decoding as $M$ goes larger, which means our code with high code rate can work efficiently with the help of multiple reads.

### 3.5.4 Other Similar Model

From table 3.1 we design a simplied channel model shown in Fig. 3.8 where $P_{AC} = P_{CG} = P_{TG} = P_{AT} = P_{AG} = \lambda * P_{CT}$. Simulation results in fig. 3.7d shows that our decoding methods can work effectively on different $\lambda$ and they work better when $\lambda$ gets larger.

### 3.5.5 Density Evolution

We first compare the largest capacity of the four methods and the baseline method by using density evolution algorithm under the (3,6) LDPC code. The result is shown in table 3.3. We can see that our methods have considerable improvement compare to the baseline method.

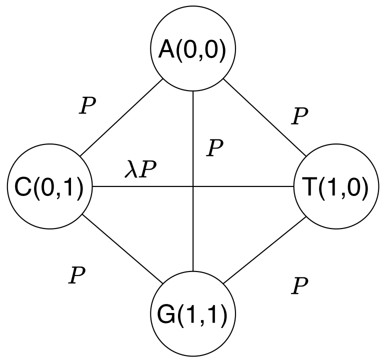


Figure 3.8: Error probabilities in the hard decoding nanopore channel.

Table 3.3: Density evolution comparison between the four methods with (3,6) code.

| | Baseline | Method 1 | Method 2 | Method 3 | Method 4 |
|---|---|---|---|---|---|
| Cap | 0.7684 | 0.6792 | 0.6460 | 0.6044 | 0.5774 |
| $\sigma$ | 0.3493 | 0.2638 | 0.2260 | 0.1727 | 0.1340 |

Then we pick some irregular LDPC codes with different code rates for our methods. The result is shown in table 3.4.

We implement the our density evolution methods based on [1]. The code quantifies all the pdf functions into an array with 6001 elements ranging from $-30$ to 30.

To find the optimized distributions, we first use exhaust search to find some good initiations. Then use the optimization method introduce by [97] to further optimize those distributions. Table 3.4 shows the capacity-approaching distributions under different rates and distributions. $R$ is the code rate and $\sigma = 1 - \frac{R}{C}$ [112] is the rate-capacity ratio that shows how close the rate close to the capacity.

The results show that our methods can reach the same threshold which is also closed to the channel capacity.

To see how the extrinsic information work during the density evolution, we track the $\bar{f} = \int_{-\infty}^{0} f_v^{iter}(\tau)d\tau$ during each iteration of the algorithm for method 3 and the baseline method. which is shown in Fig 3.9. We can see that Channel 1 of baseline stops with a high $\bar{f}$.

Table 3.4: GOOD DEGREE DISTRIBUTION PAIRS WITH MAXIMUM VARIABLE NODE DEGREES 15, 20, 30, AND 50. FOR EACH DEGREE DISTRIBUTION PAIR THE CODE RATE, THE CAPACITY, THE VARIANCE PARAMETER *var* ,THE CODING TYPE(i.e, HARD CODING OR SOFT CODING), AND THE CORRESPONDING $\sigma$

| $\lambda_2$ | 0.2340 | 0.2558 | 0.036 | 0.4222 | 0.4460 | 0.4627 |
|---|---|---|---|---|---|---|
| $\lambda_3$ | 0.3051 | | 0.2625 | | | |
| $\lambda_4$ | | | 0.0096 | | | |
| $\lambda_5$ | | | | 0.0355 | 0.0164 | |
| $\lambda_9$ | | | | 0.1286 | | |
| $\lambda_{11}$ | | | 0.2346 | | | |
| $\lambda_{19}$ | 0.1570 | | | | | |
| $\lambda_{20}$ | 0.3038 | | | | 0.1709 | |
| $\lambda_{43}$ | | | | 0.4136 | | 0.0170 |
| $\lambda_{48}$ | | 0.4978 | | | | |
| $\lambda_{50}$ | | 0.2464 | 0.4574 | | 0.3667 | 0.5202 |
| $\rho_8$ | 0.7188 | | | 0.7188 | 0.7188 | 0.7188 |
| $\rho_9$ | 0.2812 | | | 0.2812 | 0.2812 | 0.2812 |
| $\rho_{10}$ | | | | | | |
| $\rho_{28}$ | | 1 | | | | |
| $\rho_{58}$ | | | 1 | | | |
| Rate | 0.5 | 0.7506 | 0.8754 | 0.5 | 0.5 | 0.5 |
| Cap | 0.5350 | 0.7687 | 0.8838 | 0.5120 | 0.5120 | 0.5120 |
| $\sigma$ | 0.0684 | 0.0235 | 0.0095 | 0.0234 | 0.0234 | 0.0234 |
| *var* | 0.58 | 0.4 | 0.28 | 1.27 | 1.27 | 1.27 |
| Type | hard | hard | hard | soft | soft | soft |
| | method 3 | method 3 | method 3 | method 2 | method 3 | method 4 |

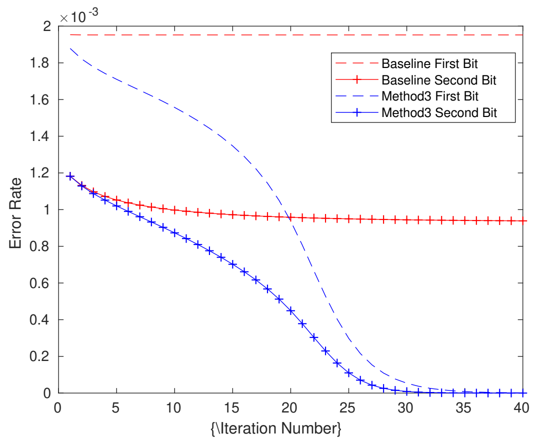


Figure 3.9: $\bar{f} = \int_{-\infty}^{0} f_v^{iter}(\tau)d\tau$ during each iteration of density evolution of Algorithm 3 and baseline algorithm

Table 3.5: Graph Circle Number Comparison using a (3,6) parity check matrix with n = 1000.

| Circle Number | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
|---|---|---|---|---|---|---|---|---|---|
| Original Graph | 0 | 2124 | 25882 | 245094 | 1811744 | 7087066 | 8105732 | 833956 | 2402 |
| Joint Graph | 72 | 2996 | 41836 | 443792 | 3479830 | 13117742 | 12978814 | 2045578 | 41340 |

Method 3's $\bar{f}$ keep decreasing with the 'help' of channel 2.

### 3.5.6 Joint Coding

The joint code has a longer codeword, however suffer from more circles under our methods due to the information exchange. We add the imaginary check node in the parity check matrix H and find all the circles in the corresponding tanner graph as shown in table 3.5 . We can see an explicit growing of the circle number comparing to the original graph.

The simulation results show that when the variance factor is from 0.85 to 0.95, joint encoding is worse than separate encoding by a factor of 4.9371X to 1.4271X in (3,6) soft Coding.

## 3.6 Conclusion

In conclusion, we present binary LDPC codes for DNA storage to combat asymmetric sequencing errors. Our decoding algorithms utilize the extrinsic information exchange to accommodate the dependency of the errors of the two bits in a nucleotide symbol. We demonstrate the desirable performance of our codes in terms of bit error rate and decoding speed.

We modify the density evolution for our decoding methods and prove that our methods can approach the capacity of the nanopore channel models.

We use multiple reads to improve our codes with higher code rate.

Our work only considers the substitution errors because we assume that the current alignment technique can solve the deletion error perfect. However, those technique all works for hard information. One future work is to consider soft information alignment for soft decoding.

# Chapter 4

# Constrained Codes for Nanopore-based DNA Storage

## 4.1 Introduction

DNA storage has been a promising alternative storage method in recent years following the works of Church et al. [21] and Goldman et al. [43]. Among the DNA sequencing (reading) technologies, Oxford Nanopore sequencing enables portable and affordable applications [123, 70, 19]. When a DNA sequence passes the nanopore sequencer, an analog current signal is generated, and an estimation algorithm (called base-calling) is used to detect the associated DNA sequence. However, effective coding methods are essential in order to combat the high rate of errors in nanopore. In this paper, a source of synchronization errors is identified, which leads to potential missing symbols and even error propagation. Accordingly, constrained codes are developed to alleviate such errors.

**Related work.** There are several pioneering DNA storage techniques created based on nanopore sequencing. For example, Yazdi et al. [123] create a random access DNA-based data storage system using error-prone nanopore sequencers. Lopez et al. [70] develop algorithms that increase the throughput of nanopore sequencing and decode 1.67 MB of information in DNA. Organick et al. [90] use nanopore sequencing to develop an applicable and large-scale system that stores over 200 MB of data in DNA. Chen et al. [19] develop nanopore-based DNA hard drives which can store, operate and read data in the changeable three-dimensional structure of DNA.

In order to improve the reliability of the DNA storage process, several coding methods have been proposed, among which the constrained code is an important class. Chee and Ling [18] study the GC-content constraint, meaning the fraction of G and C nucleotides of DNA codewords should be approximately 50% to prevent insertion and deletion errors in polymerase chain reaction (PCR) [76]. Immink and Cai [54] focus on the homopolymer (or the run-length) constraint, which does not allow consecutive identical nucleotides in DNA codewords to prevent sequencing errors [21, 30]. There are some works that have been done to satisfy both of those two constraints. The work of [109] calculates the optimal average bits that can be stored by one nucleotide under both constraints. It also implements its own code which satisfies the those two constraints. The work of [120] develops a GC-balanced run-length limited code that uses both short constrained sequences and an encoding algorithm to map information data to long DNA sequences for the constraints. Limbachiya et al. [68] develop an error correcting code based on greedy exhaustive search and Reed–Solomon code for those two constraints. Nguyen et al. [84] also develop coding techniques that account the two constraints but with a higher coding rate than the existing methods.

Our paper studies a different constraint, which is posed to reduce the potential synchronization errors during identification of the sequencing signals. Figure 4.1 shows a simulated current signal of a DNA sequence using a $k$-mer model [113], where every $k = 6$ DNA bases (called a 6-mer) generates one fragment of current signal. The length of a fragment is a ran-

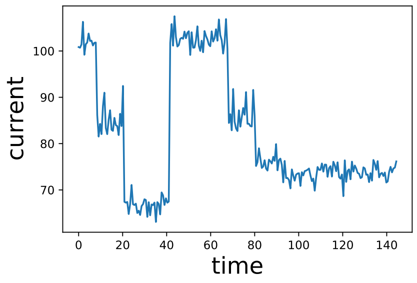


Figure 4.1: Current signal of the DNA sequence 'ACCAGATTGGGGGGGG'. The signal is generated from the Gaussian model in [113]. Each 6-mer corresponds to a segment of random time duration and current.

dom variable [1]. In our example, ACCAGA corresponds to the first fragment for time from 0 to 11, CCAGAT corresponds to the second fragment from 12 to 19, and so on. Within a fragment, all current values are independent and identically distributed Gaussian random variables, whose mean and variance depend on the 6-mer. Synchronization errors happen when the current difference between adjacent 6-mers is small. For example, the mean current values of CAGATT and AGATTG are 66.6 and 66.5, shown for time from 20 to 42. It is difficult to distinguish the two fragments, leading to potential synchronization errors during base-calling. For example, we may miss AGATTG. Note that two adjacent 6-mers overlap by 5 symbols. When some 6-mer is missing, the estimation of the following multiple 6-mers are subject to errors, leading to possible error propagation.

In order to avoid the above synchronization errors, we propose constrained codes where adjacent 6-mers are allowed only if the difference of their Gaussian means are above a threshold. As a special case, our constraint includes the homopolymer constraint, because consecutive 6-mers of the same base have the same Gaussian mean. For example, in Figure 4.1, the homopolymer GGGGGGGG generates indistinguishable signals from time 78 to 145.

The main contributions of this paper are as follows. First, we define our constraints on DNA

[1] In practice, the current signal can be first segmented into events, where several events correspond to one $k$-mer [27]. In this case, one can view each signal value for one unit of time in Figure 4.1 as the average current of one event.

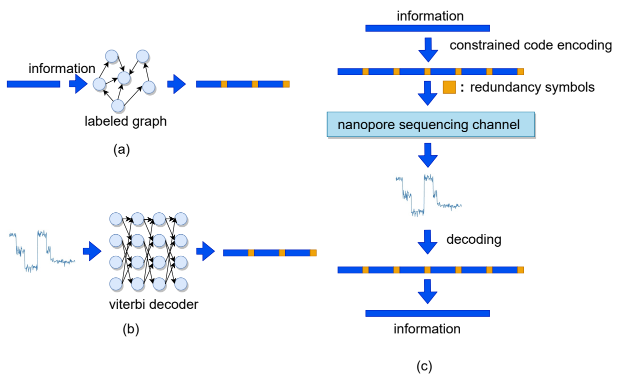


Figure 4.2: a constrained code system for synchronization error correction. a: DNA information is encoded by label graph of the constrained encoder to the constrained codeword. b: current information from nanopore sequencing is decode by viterbi decoder back to the codeword. c: overall encoding and decoding procedure.

sequences for the synchronization errors during nanopore sequencing. The DeepSimulator [67] data shows that by using constrained code, the deletion error decreases between 0.74% and 1.27% depends on different parameter settings. Then, based on the classical constrained coding algorithms [77], we design algorithms for the quaternary code and present methods to reduce the time and space complexity. Moreover, we develop a decoder based on Viterbi algorithm [35] and Needleman–Wunsch algorithm [82] for the constrained code that allows opportunistic correction of additive Gaussian noise and deletions. Simulation results show that our decoder at the cost of 16.7% in the coding rate achieves less than $10^{-4}$ when deletion rate is less than 5% and the Gaussian noise factor is 0.2.

Figure 4.2 shows the overall encoding and decoding procedure of constrained code. First, the information will be encoded by constrained code labeled graph. Then, the output current information from nanopore channel will be be decoded by Viterbi Algorithm to the original encoded constrained code. Last, the encoded constrained code which will be further decoded to the original information.

The rest of the paper is organized as follows. In Section 4.2, we introduce the constrained code for nanopore sequencing, construct efficient encoders and decoder for error correction of the constrained code. In section 4.3, we present simulation and experiment results. Con-

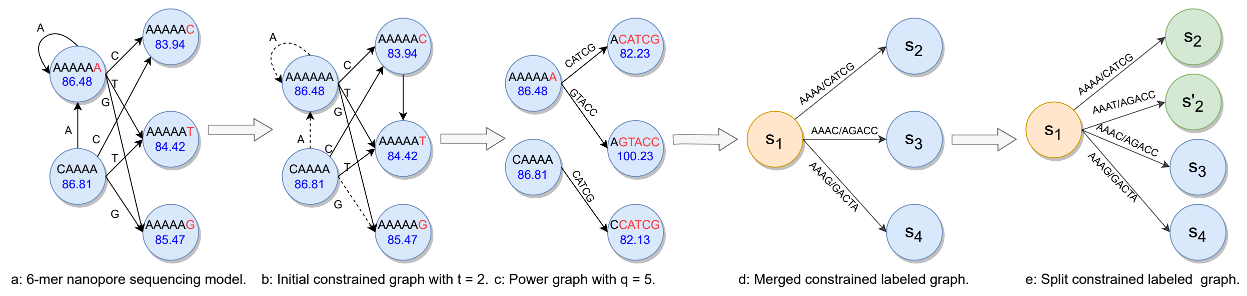

AAAAAA 86.48
AAAAAC 83.94
AAAAAT 84.42
AAAAAG 85.47
CAAAA 86.81
ACATCG 82.23
AGTACC 100.23
CCATCG 82.13
CATCG
GTACC
AAAA/CATCG
AAAC/AGACC
AAAG/GACTA
AAAT/AGACC
$s_1$
$s_2$
$s'_2$
$s_3$
$s_4$
a: 6-mer nanopore sequencing model.
b: Initial constrained graph with t = 2.
c: Power graph with q = 5.
d: Merged constrained labeled graph.
e: Split constrained labeled graph.

Figure 4.3: Procedure of generating constrained code encoder for nanopore sequencing when $t = 2, p/q = 4/5$. In Parts a, b, and c, the edge label denotes codeword. In Parts d and e, the edge label denotes information/codeword.

clusions are made in Section 4.5.

**Notation.** Let $\{A, C, G, T\}^n$ denote all DNA sequences of length $n$. We denote by $\{A, C, G, T\}^*$ all possible DNA sequences whose length is unspecified. We use bold font, for example $\mathbf{G}$, to denote a graph. The number of states (vertices) and the number of edges of the graph is denoted by $N_S(\mathbf{G}), N_E(\mathbf{G})$, respectively. When the graph is clear from the context, they are simply written as $N_S, N_E$.

## 4.2 Models and Methods

### 4.2.1 Synchronization Errors and the Threshold Constraint

Our nanopore sequencing model is simplified from the $k$-mer model of [113], where $k$ is set to be 6. A DNA sequence $x = (x_1, x_2, ..., x_n) \in \{\text{A,C,G,T}\}^n$ generates a current value from every $k$ nucleotides. The current signal is denoted by $y = (y_1, y_2, ..., y_{n-5}) \in \mathbb{R}^{n-5}$, where $y_i$ is a realization of a Gaussian random variable $Y_i \sim \mathcal{N}(\mu_{u_i}, \sigma^2_{u_i})$, whose mean $\mu_{u_i}$ and variance $\sigma^2_{u_i}$ are determined by the $k$-mer $u_i \triangleq (x_i, x_{i+1}, ..., x_{i+5})$, $1 \leq i \leq n-5$. In the model, each $y_i$ equals the average of the segment of the signal for the $k$-mer $u_i$ (see Figure 4.1). We say that the current signal has additive Gaussian noise. There are two types of *deletion errors* such that $y_i$ is absent from $y$. First, the segment corresponding to $y_i$ is skipped entirely, which is called a *skipping error*. Assume the deletion errors are independent and identically distributed for each $k$-mer, and $\Pr(\text{deletion}) = \epsilon$. Second, the segments corresponding to $y_{i-1}$ and $y_i$ are indistinguishable during base-calling, and erroneously detected as one fragment, which is called a *synchronization error*. For two $k$-mers $u_{i-1}$ and $u_i$, it is more likely to have a synchronization error when $|\mu_{u_i} - \mu_{u_{i-1}}|$ is smaller.

Our main goal is to reduce the number of synchronization errors by imposing the *threshold constraint* on the DNA sequence. In particular, we choose a threshold $t \geq 0$ and only allow adjacent $k$-mers such that

$$|\mu_{u_i} - \mu_{u_{i-1}}| \geq t, \forall 1 \leq i \leq n - 5. \tag{4.1}$$

**Verification of the threshold constraint.** A simple mathematical model is developed in the supplemental material to measure the synchronization error rate as a function of the threshold $t$. Below we justify the threshold constraint from two sets of data. We use DeepSimulator [67] to test the probability of deletion errors as a function of the threshold. DeepSimulator uses a deep neural network to generate the current for a given DNA sequence and then uses a nanopore base-calling algorithm (we used guppy_3.1.5 base-caller [47]) to decode the current to the DNA sequence. Finally, Deepsimulator uses minimap2 [65] to check the sequencing accuracy. In our simulation, we randomly generate 4000-long DNA sequences with several thresholds (for $t > 0$, the sequences are generated using encoding methods introduced in Section 4.2.3). We compare the average sequencing accuracy by minimap2 [65] and the average insertion, deletion, and substitution error rates by using the Needleman–Wunsch algorithm [82]. From Table 4.1 we can see that the threshold constraint reduces the average error rates.

Moreover, we tested the relation between deletion errors and adjacent $k$-mer mean difference

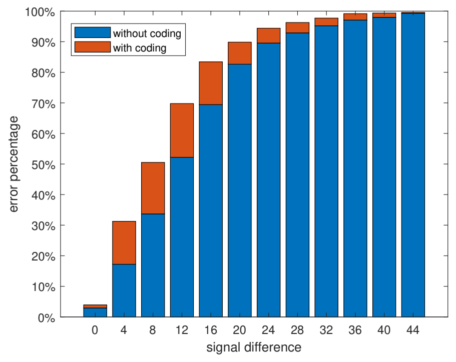


Figure 4.4: Deletion error rate improvement from threshold constraint on human genome data. Horizontal axis represents the adjacent $k$-mer signal mean difference or the threshold $t$. Blue bars show the cumulative percentage of adjacent $k$-mers with the give $t$. Yellow bars represent the cumulative percentage of errors with the corresponding $t$.

with the NA12878 human genome reference on the Oxford Nanopore MinION using 1D ligation kits (450bp/s) with R9.4 flow cells, which is also used by [67]. Figure 4.4 shows our simulation on the human genome data. We first get the distribution of the adjacent $k$-mer mean from the human DNA sequence of [67], represented cumulatively by the blue bars. For example, 31% of the adjacent mean difference is no less than 8. Then, we use guppy_3.1.5 base-caller to get the estimated DNA sequence from the current signal of [67], and run Needleman-Wunsch algorithm to find deletions in the estimated sequence. The cumulative deletion percentage is shown by the yellow bars. For instance, 50% of deletions happen when the $k$-mer mean difference is smaller than or equal to 8. Notice that when designing a DNA sequence for storage, unlike human genome, we can choose any sequence as desired. Therefore, from Figure 4.4, it can be observed that a significant percentage of deletions can be avoided if we apply the threshold constraint with a small constant $t$. We first check the raw current data from human DNA of [67] and get the distribution of current difference between every two 6-mers which is represented by the blue bar. For example, the 31% of the signal difference is smaller than 8. Then, we use Needleman-Wunsch algorithm to check deletions of the output DNA string from the basecaller. The deletion result is shown from the yellow bar, which means that 50% of deletions happen when the signal difference is smaller than 8.

### 4.2.2 Constrained Codes for Nanopore Sequencing

To effectively convert arbitrary information into DNA sequences satisfying the threshold constraint, we adopt the concepts of constrained codes [77]. We first introduce the representation of DNA sequences by graphs and calculate the coding capacity, and then present our encoder and decoder constructions.

The possible transitions between adjacent $k$-mers in a DNA sequence can be naturally represented by a *labeled graph*, as partially shown in Figure 4.3.a. There are a total of $N_S = 4^6$ states (or vertices) and $N_E = 4^7$ directed edges, and each state corresponds to a 6-mer. Every state has four outgoing edges that are linked to the next possible 6-mers. Thus, state $u = (x_1, x_2, ..., x_6)$ has an outgoing edge $e$ linked to state $(x_2, x_3, ..., x_7)$, for any $x_7 \in$ {A,C,G,T}, and the edge label is defined as $L(e) = x_7$. To define the constrained code, we also include the mean of the current signal $\mu_u$ generated by nanopore sequencing for each state $u$ in Figure 4.3.a.

As mentioned, if two adjacent 6-mers generate similar current values, a synchronization error may occur. We thus define *forbidden words* as adjacent 6-mers whose current mean difference is less than a given threshold $t$:

$$\begin{aligned}\mathcal{C}_f \triangleq \{&(x_1, x_2, \ldots, x_7) : |\mu_u - \mu_v| < t,\\ &u = (x_1, \ldots, x_6), v = (x_2, \ldots, x_7)\}.\end{aligned} \tag{4.2}$$

Notice that homopolymers such as AAAAAAA (seven A's) are automatically forbidden by the above definition.

Our *constrained code* $\mathcal{C} \subseteq \{A, C, G, T\}^*$ is defined to be all DNA sequences that do not contain any forbidden word as a subsequence. The constrained code can be represented by the labeled graph by removing the edges between states $u$ and $v$ when $|\mu_u - \mu_v| < t$. Let us denote the corresponding graph by $\mathbf{G}$. Figure 4.3.b shows an example when we set $t = 2$ and remove three edges. All allowed DNA sequences (codewords) of the constrained code can be generated by reading the edge labels of each possible path in $\mathbf{G}$.

Next, we quantify the highest possible coding efficiency of a constrained code. Let $\mathcal{C}(\ell)$ be the set of codewords of $\mathcal{C}$ whose length is $\ell$, $\ell \in \mathbb{N}$. Let $A_{\mathbf{G}}$ be the adjacency matrix of size $N_S \times N_S$, whose $(u, v)$-th entry equals to the number of edges from state $u$ to state $v$ for any labeled graph $\mathbf{G}$.

The *Shannon capacity* of the constrained code $\mathcal{C}$ is define as the asymptotic number of information symbols per codeword symbol:

$$cap(\mathcal{C}) = \limsup_{\ell \to \infty} \frac{\log_4 |\mathcal{C}(\ell)|}{\ell}.$$

It is well known that the capacity can be computed from the adjacency matrix [77] by

$$cap(\mathcal{C}) = \log_4 \lambda(A_{\mathbf{G}}), \tag{4.3}$$

where $\lambda(A_{\mathbf{G}})$ is the largest eigenvalue of $A_{\mathbf{G}}$ for a graph $\mathbf{G}$ that has distinct labels for the outgoing edges of each state. It is apparent our graph $\mathbf{G}$ satisfies the above condition.

Moreover, we can in fact find a connected subgraph of $\mathbf{G}$ whose capacity is equal to $cap(\mathcal{C})$. Hence from here on, $\mathbf{G}$ simply denotes this connected graph. For the examples we tested, the number of states in the connected graph and the capacity are listed in the second and the last columns of Table 4.2.

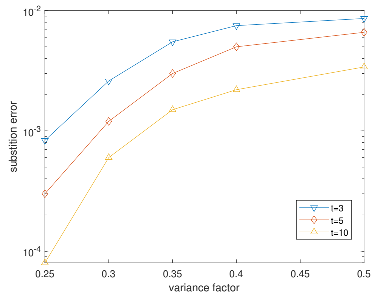
It can be seen that for a larger threshold $t$, more edges and paths (words) are removed, hence the capacity is reduced. In Table 1 of supplemental material, we show the trade-off between the synchronization error rate and the capacity.

Table 4.1: DeepSimulator result for DNA sequence with threshold $t = 0, 3, 5, 10$.

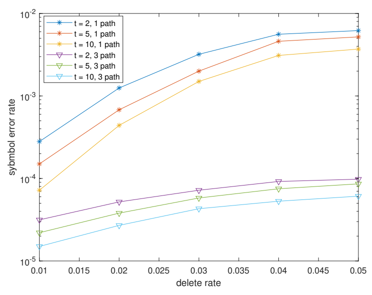
| | $t = 0$ | $t = 3$ | $t = 5$ | $t = 10$ |
|---|---|---|---|---|
| Base-calling accuracy | 97.44% | 98.28% | 98.67% | 98.98% |
| Deletion error rate | 1.76% | 1.02% | 0.75% | 0.59% |
| Insertion error rate | 0.34% | 0.33% | 0.28% | 0.12% |
| Substitution error rate | 2.15% | 1.59% | 1.11% | 0.77% |

Table 4.2: The number of states of the labeled graphs $\mathbf{G}$, $\mathbf{G}_m$, $\mathbf{G}_s$, for $t = 3, 5, 10$. The chosen code rate $p/q$ is also listed.

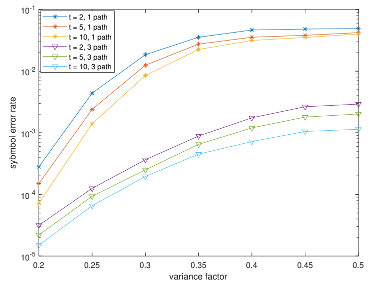
| signal threshold | initial | merged | split | $p/q$ | symbol capacity |
|---|---|---|---|---|---|
| $t = 3$ | 2952 | 185 | 185 | 5/6 | 0.9166 |
| $t = 5$ | 2021 | 116 | 167 | 4/5 | 0.8494 |
| $t = 10$ | 751 | 98 | 157 | 3/5 | 0.6338 |



(a)



(b)



(c)

Figure 4.5: Symbol error rates after Viterbi decoder for different constraint threshold $t$. (a) Symbol error rate using single-path Viterbi algorithm. The variance factor of the additive Gaussian noise ranges from 0.25 to 0.5, and there are no deletions. (b) Symbol error rate using single-path and 3-path Viterbi algorithm. The variance factor equals 0.2 and the deletion rate varies from 1% to 5%. (c) Symbol error rate using single-path and 3-path Viterbi algorithm. The deletion rate equals 1% and variance factor varies from 0.2 to 0.5.

### 4.2.3 Encoder

While the paths of the labeled graph represent all possible codewords, an efficient encoder/decoder is to be constructed. We apply the methods in[77] and obtain a finite-state encoder with a fixed rate $p/q$ that is arbitrarily close to the capacity, where $p, q$ are integers such that $p/q < cap(\mathcal{C})$. Our encoder is built by creating another labeled graph through two operations: graph power and state splitting. In the resulting graph, each edge has an output label (codeword) of length $q$, and from each state there are $4^p$ edges, corresponding to an input label (information) of length $p$. Moreover we present how to generate a graph representation of our constrained code with a small number of states. In all the above graph operations, the original constraints of forbidden words are still maintained.

**Graph power.** Once the code rate $p/q$ and the reduced labeled graph $\mathbf{G}$ have been decided upon, the $q$-th power of $\mathbf{G}$, denoted by $\mathbf{G}^q$, is generated. It has the same states as $\mathbf{G}$, but the edges from $u$ to $v$ correspond to all $q$-hop paths in $\mathbf{G}$ from $u$ to $v$. Each edge label becomes length $q$ and corresponds to the $q$-hop path label. The adjacency matrix can be obtained from the matrix power $A_{\mathbf{G}}^q$. Figure 4.3.c shows an example of $\mathbf{G}^q$. It can be seen that the codewords of $\mathbf{G}$ and $\mathbf{G}^q$ are identical.

Although the matrix power calculation is mathematically simple, two issues arise when the power $q$ and the number of states $N_S = N_S(\mathbf{G}) = N_S(\mathbf{G}^q)$ are large: time and space complexity. The time complexity would be $O((q-1)N_S^3)$ if we directly use the matrix multiplication. We can reduce it to no more than $O(N_S + N_E(\mathbf{G})) = O(N_S)$ by using the Breadth-First-Search (BFS) algorithm with fixed $q$ hops for each state, where $N_E(\mathbf{G}) \leq 4N_S$.

However, the space complexity to store the powered graph is $O(N_S + N_E(\mathbf{G}^q))$, where $N_E(\mathbf{G}^q)$ can be close to $4^q N_s$ for small $t$. For example, when $q = 8$, there are more than $10^8$ edges in the powered graph. We propose the following 3 methods to alleviate the memory problem. First, use efficient representation for the edges, e.g., the nucleotide symbol is stored as 2 bits instead of one character. Second, partition the states into subsets, and store the edges from one subset to another as a block in main storage. When processing the graph, handle the states according to the subsets. It must be noted that the graph partition increases the running time due to the lower I/O speed of main storage. Third, reduce the graph size by state merging explained below.

**State merging.** State merging algorithm [77] reduces the graph by merging the states of $\mathbf{G}^q$ such that the resulting codewords are a subset of the original set of codewords (hence the constraints are maintained), and the code rate $p/q$ is still achievable.

The *follower set* $\mathcal{F}_{\mathbf{G}^q}(u)$ is defined as the set of all the words that can be generated starting from state $u$ in $\mathbf{G}^q$.

We merge state $u'$ into state $u$ if they satisfy $\mathcal{F}_{\mathbf{G}^q}(u) \subseteq \mathcal{F}_{\mathbf{G}^q}(u')$ and have identical approximate eigenvector (see details in [77]). The merging process is to remove all the outgoing edges of $u'$, redirect all the incoming edges of $u'$ to $u$, and then remove state $u'$.

Denote the graph after state merging by $\mathbf{G}_m$. See Figure 4.3.d for an example. The third column of Table 4.2 shows the result of state merging. We can see that the state number $N_S$ dramatically decreases.

**State splitting.** Up to now, if each state in $\mathbf{G}_m$ has $4^p$ or more outgoing edges, we are done with graph transformations. Otherwise, we apply the state splitting algorithm [77] to get the graph $\mathbf{G}_s$ with minimum out-degree $4^p$ and then attain the desired $p/q$ encoder. The high-level idea is to find certain states, duplicate the states and their incoming edges, and then split their outgoing edges among the duplicated states. Figure 4.3.e shows an example of how state splitting increases the out-degree of some states: After we split $S_2$ into two new states in green, the out-degree of its parent state $S_1$ increases by one.

In each iteration of splitting, one state is picked and split. The total number of states $N_S$

is hence increased by one. There are usually multiple candidate states for splitting in each iteration, which may result in different numbers of states and different decoding delay.

However, since the aim of splitting one state is to increase the out-degrees of its parents, if these degrees are already larger than or equal to $4^p$, then the split is unnecessary. Due to this reason, we improve the state splitting algorithm by checking the out-degrees of the parents. Simulation results show that the improved algorithm reduces around 20% for the splitting iterations as well as the number of additional states. The fourth column of Table 4.2 shows that the number of states $N_S$ does not have a substantial increase after the improved splitting algorithm.

**Encoding.** We are now ready to build the encoder. We remove edges in $\mathbf{G}_s$ such that the out-degree of every state is exactly $4^p$. The $q$-symbol label of each edge is now called the *output label*, corresponding to a codeword. We can simply construct the encoding graph by adding one $p$-symbol *input label* to each edge, representing the input information, as shown in Figure 4.3.e. For example, the information sequence AAAA will be encoded into CATCG if the current state is $S_1$, and then the next state will be $S_2$. Hence, we have a rate $p/q$ finite-state encoder. The encoding can then be accomplished as follows.

1. Initialize an arbitrary state $u_0$ as the current state $u$. Initialize the stage number to be $i = 0$.

2. Get $p$-symbol subsequence $s = (s_{ip+1}, s_{ip+2}, \dots, s_{ip+p})$ from the input information sequence, and find the edge $e$ from $u$ with the input label $s$. Append the $q$-symbol output label of $e$ to the output codeword. Update the current state $u$ as the destination state of $e$. Increase $i$ by one.

3. Repeat Step 2 until the input information is exhausted.

The required memory size is $2(p + q)4^p N_S$ bits, which is around hundreds of KBs in our experiment.

### 4.2.4 Viterbi Decoder

Assume there are additive Gaussian noise and deletions (including residual synchronization errors after the constrained code) in the current signal from nanopore sequencing. Even though constrained code is not designed for error correction, we design a two-step decoder to exploit the code redundancy and opportunistically correct errors. Specifically, the current signal can be viewed as a realization of a hidden Markov model for which the maximum a posteriori codeword and the hidden state of the labeled graph can be found by Viterbi algorithm.

Given the current signal, we assign scores to each possible state transition ($q$ additional codeword symbols), and choose the overall codeword with the optimal score. To avoid exponential increase in complexity due to the large number of possibilities, the decoder consists of two dynamic programming steps. First, we use Needleman–Wunsch algorithm [82] to find the highest score for each edge at every stage where potential deletions are taken into consideration. Second, Viterbi algorithm [35] is used to obtain the best path given the edge scores. Information is then decoded from the best path.

**Viterbi algorithm.** We apply Viterbi algorithm to find the best path for the given edge scores.

Figure 4.6 shows the stages in Viterbi algorithm. Each stage has $N_S$ states and each state has $4^p$ outgoing edges to the next stage. The edges between two stages correspond to the same transitions as the labeld graph $\mathbf{G}_s$. Each edge is associated with a score corresponding to the posterior probability to be discussed later. For state $S_i$ at a stage, we compare the scores of all its incoming paths and keep the one with the highest score.

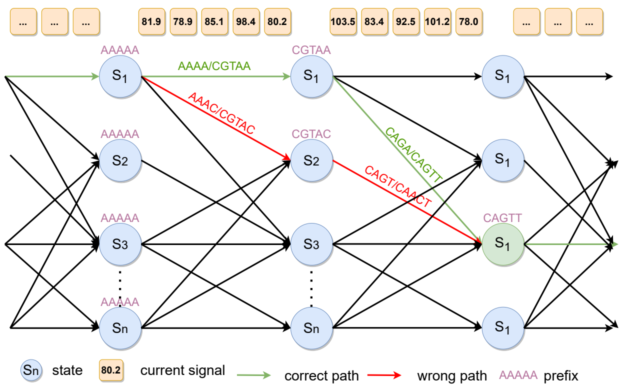


Figure 4.6: Viterbi algorithm. The wrong path converges back to the correct state after two stages.

Errors might happen during this process when a wrong edge has a higher score than the correct edge at some stage. However, after a few stages the wrong path is likely to accumulate more and more errors unless it converges to the correct state. When the signal noise is moderate, simulation confirms that Viterbi algorithm does not propagate the error. An example is shown in Figure 4.6. In Stage $i+2$, the optimal edge into State $S_3$ is the wrong (red) one due to an erroneously high score of the red edge in Stage $i+1$. However, the correct (green) edge in Stage $i+3$ is found regardless of the previous errors.

We note that Viterbi algorithm has been used in several works for nanopore base-calling. Timp et al. [115] uses Viterbi algorithm to decode the current signal from 3-base-pair-resolution nanopore electrical measurement. The work by Chandak et al. [15] uses viterbi algorithm as an error correcting decoder which enables the basecaller to directly work on the soft information. David et al. [27] builds a basecaller where Viterbi decoding is applied to find the 6-mer as the hidden state.

**Needleman–Wunsch algorithm.** Viterbi algorithm requires a score for each edge, for instance hamming distance under the hard decoding model. To correct deletion errors, we apply Needleman–Wunsch algorithm [82], which is a global sequence alignment algorithm. To accommodate the soft information (analog current signal), we design the scoring system based on posterior probabilities.

Consider one edge $e$ in Figure 4.6 between two stages. Let $\hat{x} \in \{A, C, G, T\}$ be the potential present codeword symbol to be scored. Let $y \in \mathbb{R}$ denote the current signal for the present digit. Notice that due to possible deletions, the present digit can vary for different edges. Denote by "prefix" the previous 5 codeword symbols, only with which can we decide the Gaussian distribution of the signal under our 6-mer model. Initially, prefix is the last 5 codeword symbols corresponding to the source state of $e$. After that, prefix is updated by one symbol at a time according to the alignment.

The posterior probabilities without and with deletion are then

$$\begin{aligned}&\Pr(\hat{x},\ \text{no del}|y, \text{prefix})\\ =&\frac{f(y|\text{prefix}, \hat{x})P(\hat{x}|\text{prefix})(1-\epsilon)}{f(y|\text{prefix})},\end{aligned} \tag{4.4}$$

$$\Pr(\hat{x}, \text{del}|y, \text{prefix}) = \epsilon P(\hat{x}|\text{prefix}), \tag{4.5}$$

where $\epsilon$ is the deletion error rate, $f(y|\text{prefix}, \hat{x})$ is the Gaussian density function for the 6-mer $(\text{prefix}, \hat{x})$, and $P(\hat{x}|\text{prefix})$ is the transition probability of the present digit computed from the labeled graph $\mathbf{G}_s$, assuming all $4^p$ out-going edges from one state are equally likely.

Figure 4.7 shows the Needleman–Wunsch algorithm. For $0 \le j \le i \le q-1$, the $(i, j)$-th score $sc(i, j)$ in the diagram can be calculated from its diagonal neighbor's score $sc(i-1, j-1)$ and the probability in (4.4), or its vertical neighbor's score $sc(i-1, j)$ and the probability in (4.5). If the former score is larger, we say there is no deletion error. Otherwise, there is one deletion. Note that the vertical neighbor does not exist if $i = j$. The yellow path in Figure 4.7 shows the alignment with the largest score. We can see that there is one deletion error at the last digit.

To simplify the algorithm, we observe that $1 - \epsilon \approx 1$, and $P(\hat{x}|\text{prefix}) \approx 1/4$ can be both

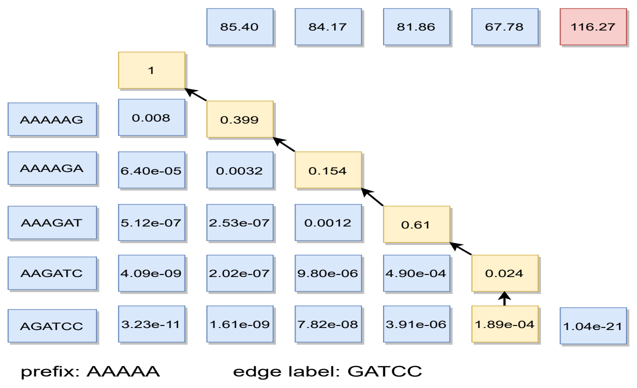


Figure 4.7: Needleman–Wunsch algorithm for output edge label GATCC with prefix AAAAA. The numbers on the top represents the current signal. The DNA sequences on the left means the current 6-mer. The rows and columns are indexed from 0 to $q-1=5$.

discarded in (4.4) and (4.5). In particular, the deletion score (4.5) becomes $\epsilon$, and the no-deletion score (4.4) becomes $\frac{f(y|\text{prefix},\hat{x})}{f(y|\text{prefix})}$. Denote by $p_{i,j}$ the no-deletion score computed from $\frac{f(y|\text{prefix},\hat{x})}{f(y|\text{prefix})}$ on the $(i,j)$-th entry in Figure 4.7. The following theorem presents a new score and the proof is shown in supplemental material.

**Theorem 4.1.** Define a new score $sc'(i,j)$ as:

$$
\begin{aligned}
& sc(i,j) \\
&= \begin{cases} sc(i,0)=\epsilon^i, & 0 \le i \le q-1, j=0, \\ sc(i,i), & i=j, \\ \max\{sc'(i-1,j-1)p_{i,j}, sc'(i-1,j)\}, & i>j. \end{cases}
\end{aligned}
\tag{4.6}
$$

Then under the approximations $1-\epsilon \approx 1$, and $P(\hat{x}|\text{prefix}) \approx 1/4$, the score satisfies

$$
sc(i,j) = sc'(i,j)\epsilon^{i-j}. \tag{4.7}
$$

Since the highest edge score is chosen among the last row in Figure 4.7, and $\epsilon$ is multiplied $i$ times in the $(q-1-i)$-th column and the last row, for $0 \le i \le q-1$, we can just update the new score $sc'(i,j)$ according to Theorem 4.1 and multiply $\epsilon^{q-1-i}$ in the last row. The powers of $\epsilon$ can be pre-computed and hence this simplified algorithm roughly halves the number of multiplications.

**Multi-path Viterbi decoder.** When the variance of the additive Gaussian noise becomes larger, simulation shows that the error rate will dramatically increase even with a small deletion rate. This is because the Gaussian noise causes false positive detection of deletions in the Needleman-Wunsch algorithm. To solve this issue, we improve the Viterbi decoder by saving multiple paths instead of one optimal path. The method is also know as list decoding, which is proven to be effective to improve the accuracy of Viterbi Algorithm [100, 85].

**Decoding.** Our decoder first obtains the maximum a posteriori codeword using dynamic programming explained above. After that, the input/output labels on the optimal path are read out as information/codeword.

1. For each stage, run Needleman–Wunsch algorithm to obtain the edge scores, and Viterbi algorithm to obtain the incoming edge with the highest score for every state, given the current signal $y = (y_1, y_2, \dots)$.

2. In the final stage, select the state with the highest score. Back track across all stages to find the optimal path.

3. For each edge on the optimal path, obtain the codeword $(x_{jq+1}, x_{jq+2} \dots, x_{jq+q})$ and the information $(i_{jp+1}, i_{jp+2}, \dots, i_{jp+p})$, for Stage $j = 0, 1, \dots$.

Besides Viterbi decoder, a direct decoder with lower complexity can be applied to decode constrained codes, where at each stage the $q$-symbol codeword is directly decided by the $q$-symbol current signal. However, to uniquely determine the information, some delay may be introduced due to state splitting. This is captured by the notation of *anticipation* and methods to decrease anticipation are discussed in supplemental material.

## 4.3 Accuracy of Viterbi Decoder

We test the Viterbi decoder under additive Gaussian noise and random deletion errors. Here, deletion errors consists of mostly skipping errors but also the residual synchronization errors after constrained coding, and are assumed to be independent and identically distributed for each 6-mer. We employ the 6-mer model in [113], but control the strength of the Gaussian noise as follows. We keep the mean of the 6-mer unchanged, but multiply every variance by a constant between 0 and 1, called the *variance factor*. The variance factor can be thought as the result of increasing the read depth. In particular, let $Y \sim \mathcal{N}(\mu, \sigma^2)$ be the Gaussian random variable for some 6-mer. If the 6-mer is sequenced $M$ times independently, we get $M$ independent and identically distributed samples of $Y$, whose average follows the Gaussian distribution $\mathcal{N}(\mu, \frac{\sigma^2}{M})$. Under this simple model, the variance factor is inversely proportional to the read depth.

Our simulation results are shown in Figure 4.5. In the experiments, the constrained codes are choose from Table 4.2 and the codeword length is set to be around 500. Figure 4.5.a shows the symbol error rate with no deletion and different Gaussian variance factors. A higher threshold $t$ in (4.2) reduces the capacity, but leads to a higher redundancy and hence potentially better error correction. The result combined with Table 4.1 shows that a threshold of $t = 5$ gives almost 5X better symbol error rate and 1.36X better synchronization error rate compared to $t = 3$ at the expense of only 7.3% loss of capacity.

Figure 4.5.b shows the simulation results with deletion error rate from 1% to 5% and the variance factor 0.2. The result shows that $t = 5$ gives almost 3X better error rate compared to $t = 3$. Meanwhile, three-path Viterbi decoder improves the error rate by 10X to 100X. The gain of the multi-path decoder becomes more noticeable as the deletion rate increases.

Figure 4.5.c shows the simulation results where the deletion error rate equals 1% and the variance factor varies from 0.2 to 0.5. We can see that high variance causes very poor error rate when considering deletion errors. As discussed in Section 4.2.4, this is because high variance causes more false positive deletion errors. We can solve this issue by applying multi-path Viterbi decoder, which gives us more than 10X better error rate.

## 4.4 Discussion and Future Work

**Comparison with deletion correction codes**. General deletion correction code still has a large gap between the lower bound and the explicit constructions in terms of redundancy. As a result, we only consider the simple case where single-deletion correction codes are used. Grigory Tenengolts [114] proposes nonbinary codes that can correcting single deletion or insertion. Its best known redundancy is $\bar{n} - \log_4(\frac{q^{\bar{n}}}{q\bar{n}})$ where $\bar{n}$ is the block size of the code. Next, we construct a code with a similar rate and length as our 4/5 constrained code. Assume now we set codeword length $n = 512$ and partition it into blocks of size $\bar{n} = 16$. There will be 3 redundant symbols per block. The code rate is 81.25% which is close to our 4/5 constrained code. However, the deletion correction code can only tolerate 1 deletion in 16 symbols. Consider the raw deletion rate of 1.76% from Table 4.1. Each 16-symbol block has 2.89% chance to have more than 1 deletions, resulting in a failure of deletion correction. For the overall 32 blocks the failure rate will be over 60.9%, which is apparently unacceptable.

If we further would like to correct a substitution error, we need to use single-deletion single-substitution code. Its best known redundancy is $10\log_2(n) + 17$ [106], resulting in a rate of 78.7% for $n = 500$ which is slightly worse than our 4/5 constrained code. However, this code only corrects either one deletion or one substitution. From Table 4.1, the expected number of extra deletions between no coding and $t = 5$ is $(1.76 - 0.75)\% \times 500 = 5.05$. Therefore, constrained code is far more efficient in removing synchronization errors.

**Combination with error correction.** It should be noted that although constrained codes can lower the deletion error rate and reduce the symbol error rate using Viterbi decoder, it is not designed specifically to correct errors. In fact, error correction codes can be concatenated with constrained codes [77] to further improve the reliability of DNA storage. For example, the standard concatenation is to let the information first be encoded by an error-correction code and then a constrained code and vice versa for decoding. Moreover, for multi-path Viterbi decoder, it is beneficial to add a layer of error correction in order to select the correct path among the multiple candidates.

Given the deletion insertion and substitution rates, the optimal choice of error correction code and/or constrained code remains an interesting open problem.

## 4.5 Conclusion

In conclusion, we present constrained codes for nanopore sequencing in order to limit the synchronization errors. Our encoder improves the state splitting algorithm and achieves a relatively small number of states. Our decoding algorithm uses Viterbi algorithm based on the labeled graph of the constrained code, which makes the code capable for error correction. Simulation result shows that the constraints can reduce the synchronization errors and the proposed decoder is effective in limiting errors caused by additive Gaussian noise and deletions.

# Chapter 5

# Communication Efficient and Straggler Robust Clock Synchronization

## 5.1 introduction

Clock synchronization is broadly used in distributed systems for many network and database applications [118, 59]. To achieve the increasing performance requirements of those applications, the state-of-the-art techniques have improved the synchronization accuracy to microsecond or even nanosecond level [40, 42, 93, 66]. As a result, there is a higher requirement on speed and stabilization for the implementation of the synchronization techniques. For example, the long information transaction during the implementation will significantly affect the accuracy of the synchronization [40]. Our work, from coding theory prospective, investigate an efficient way to collect and calculate the information for the clock synchronization problem.

Most research on the clock synchronization problem in a distributed system focuses on two aspects, two-clock synchronization and multiple-clock synchronization for a whole graph consensus. The two-clock synchronization addresses the reduction of one-way delay between two servers which is raised by multiple factors such as path length, temperature and fluctuations of switch times. Several protocols and algorithms are designed for this issue. The multiple-clock synchronization uses the information from the whole network to generate *approximate global time* [61] of the system [40, 63, 72, 107, 73].

Among the latest multiple-clock synchronization algorithms, HUYGENS [40] enables delay-sensitive applications by achieving an accuracy of tens to hundreds of nanoseconds. To obtain one-way delay, or clock time discrepancy, between a pair of connected servers, HUYGENS proposes a coded probe filter to purify the training data and uses support vector machines (SVM) [86] to estimate the synchronization time between two clocks. Then, it develops a loop-wise algorithm to eliminate the asymmetric mistakes between multiple loops.

To implement HUYGENS algorithm, each server node exchanges time information with its neighbouring nodes, and then all server nodes send information to a master node who runs the synchronization algorithm. As the clocks drift constantly, it is essential to ensure timely communication between the servers and the master to guarantee the synchronization accuracy. In this work, we focus on how to optimize the information transfer in this network from coding prospective.

We summarize HUYGENS algorithm [40] as follows: The network is described by the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents server nodes, and $\mathcal{E}$ is the set of edges. Each edge is associated with a weight representing the time discrepancy, also called the edge information. Every node only knows the edge information of its incident edges. A master wants to collect all the edge information and compute a particular linear function of such information.

Based on the above algorithm, we raise a new coding problem: assuming each server can linearly code its edge information, what is the fundamental limit on the communication cost from the servers to the master, and can we achieve the optimal cost?

This problem is closely related to the network computing problem where source nodes in a directed acyclic graph generate independent messages and a single master node computes a target function of the messages [3, 4, 126, 2, 53, 46]. In general, network computing considers arbitrary graphs and arbitrary target functions. Some upper bounds are proposed based on the min-cut bound. Our clock synchronization problem can be regarded as the special case of network computing, i.e., computing linear functions by linear coding over a particular graph. Suppose there are $|\mathcal{E}|$ sources, each holds the time discrepancy. Each server node is regarded as the intermediate node in the graph that receives the information from the sources and sends information to the master. The target function of clock synchronization is a linear function of the time discrepancy. Appuswamy et al. investigate linear function computing over networks [2]. They focus on the converse bound and formulate an algebraic test to determine whether an arbitrary network can compute linear functions using linear codes.

In this work, we attempt to find the optimal solution for the clock synchronization problem. Our contributions consist of three parts. First, for the case where each time discrepancy is known by both incident servers, we propose a novel coding scheme that achieves the best rate. It is interesting that our scheme can also tolerance a straggler (slow or failed server) with no extra cost. Second, for the case where each time discrepancy is known by only one of the incident servers, we show that the optimal solution is the trivial solution, i.e., sending all the time discrepancy information to the master. At last, we design an algorithm for the rest of the scenarios. Our algorithm outperforms the trivial solution and is optimal for some cases.

Notation: We use calligraphic characters to denote sets and bold characters to denote matrix. For positive integer $N$, $[N]$ stands for the set $\{1, 2, \ldots, N\}$. For a set $\mathcal{S}$, $|\mathcal{S}|$ represents its cardinality.

## 5.2 Problem Statement

We first review the concept and HUYGENS algorithm of clock synchronization [40]. Then, we introduce our communication model for the algorithm.

### 5.2.1 Clock synchronization

HUYGENS algorithm [40] first implements the pair-wise synchronization algorithm between two neighboring server clocks. It then builds a synchronization algorithm that considers the whole network effect.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the server network with server nodes $\mathcal{V}$ and communication links (edges) $\mathcal{E}$. For convenience, every edge has a predefined direction. The edge from Node $U$ to Node $V$ is denoted as $UV$, for $U, V \in \mathcal{V}$.

In the first step of HUYGENS algorithm, the edge information, or the time discrepancy, is agreed between every pair of adjacent nodes. We assume the edge information is only known by one or both of the two incident nodes. The edge information for Edge $UV$ is denoted as $l_{UV}$. In Figure 5.1.a, for example, Servers $A$ and $B$ both believe that $B$'s time is 20 time units earlier than $A$ and the time discrepancy $l_{AB} = 20$ is only known by Node $A$ and/or Node $B$. In Figure 5.1.a, the *original time discrepancy vector*

$$\mathbf{\Delta}^P = [l_{AB}, l_{BC}, l_{CA}, l_{BD}, l_{DA}]^T = [20, -15, 5, 25, -15]^T \tag{5.1}$$

corresponds to the time discrepancy on directed edges $AB, BC, CA, BD$, and $DA$.

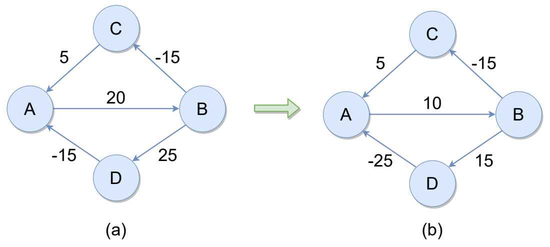


Figure 5.1: HUYGENS algorithm. The figure is redrawn from [40].

However, notice that in the loop $A \to B \to C \to A$, Server $C$ is 5 time units later than $A$ and 15 time units later than $B$. Then $B$ should be only 10 time units earlier than $A$. This means that there are 10 time units loop-wise offset surplus, due to inaccuracy of the clock measurement.

The second step of HUYGENS algorithm is to eliminate the loop-wise offset surplus in the clocks network. It calibrates the original discrepancy vector $\mathbf{\Delta}^P$ to the *final time discrepancy vector*

$$\mathbf{\Delta}^F = [10, -15, 5, 15, -25]^T$$

as shown in Figure 5.1.b. The transformation from $\mathbf{\Delta}^P$ to $\mathbf{\Delta}^F$ is explained below.

**Remark.** From the example above, we see that if the graph $\mathcal{G}$ has several connected components, then the loop-wise surplus can be eliminated for each component independently. Moreover, if some nodes are not included in any loops, they can be ignored during clock synchronization. Therefore, we assume $\mathcal{G}$ is connected and all nodes are in loops without loss of generality.

Given a server network graph $\mathcal{G}$, denote $\mathbf{A}$ as the *loop-composition matrix*. Each column of $\mathbf{A}$ represents an edge and each row represents a loop. The columns of $\mathbf{A}$ are indexed by the edges. The rows are indexed by the largest set of linearly independent loops in $\mathcal{G}$. If an edge occurs in a loop, directly or reversely, the corresponding entry in $\mathbf{A}$ is 1 or $-1$; otherwise, the entry equals 0. For a connected graph, matrix $\mathbf{A}$ has full row rank:

$$rank(A) = |\mathcal{E}| + |\mathcal{V}| - 1. \tag{5.2}$$

For example, in Figure 5.1, let the 5 columns of $\mathbf{A}$ be indexed by edges $AB, BC, CA, BD, DA$. The three loops $A \to B \to C \to A$, $A \to B \to D \to A$, and $A \to C \to B \to D \to A$ can be denoted respectively by the three rows below:

$$\begin{array}{ccccc} AB & BC & CA & BD & DA \\ \end{array}$$
$$\left(\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & -1 & -1 & 1 & 1 \end{array}\right). \tag{5.3}$$

The last row (loop) is dependent on the first two, and hence the loop-composition matrix is

$$\begin{array}{ccccc} AB & BC & CA & BD & DA \\ \end{array}$$
$$\mathbf{A} = \left(\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{array}\right). \tag{5.4}$$

The vector $\mathbf{Y} = \mathbf{A}\boldsymbol{\Delta}^P$ gives the original loop-wise surplus in each independent loop of $\mathbf{A}$. In order to apply the loop-wise correction, we look for a vector $\mathbf{N}$ which also solves $\mathbf{Y} = \mathbf{AN}$ and posit the correction to be $\boldsymbol{\Delta}^F = \boldsymbol{\Delta}^P - \mathbf{N}$. As a result, the final loop-wise surplus vector is $\mathbf{A}\boldsymbol{\Delta}^F = \mathbf{0}$. Now, $\mathbf{A}$ has full row rank, since the loops are all linearly independent. Further, since the number of linearly independent loops in $\mathcal{G}$ equals $|\mathcal{E}| - |\mathcal{V}| + 1$ which is less than $|\mathcal{E}|$, the equation $\mathbf{Y} = \mathbf{AN}$ is under-determined and has multiple solutions. We look for the minimum-norm solution since this is most likely the best explanation of the errors in the loop-wise surpluses:

$$\min_{\mathbf{N}:\mathbf{Y}=\mathbf{AN}} ||\mathbf{N}||_2. \tag{5.5}$$

Since the pseudo-inverse is well-known to give the minimum-norm solution [94]:

$$\mathbf{N} = \mathbf{A}_{right}^{-1}\mathbf{Y} = \mathbf{A}^T(\mathbf{AA}^T)^{-1}\mathbf{A}\boldsymbol{\Delta}^P, \tag{5.6}$$

where the $\mathbf{A}_{right}^{-1}$ is the right pseudo-inverse of $\mathbf{A}$[87], HUYGENS algorithm can be concluded in the following formula:

$$\boldsymbol{\Delta}^F = \boldsymbol{\Delta}^P - \mathbf{N} = (\mathbf{I} - \mathbf{A}^T(\mathbf{AA}^T)^{-1}\mathbf{A})\boldsymbol{\Delta}^P, \tag{5.7}$$

where $\mathbf{I}$ is a $|\mathcal{E}| \times |\mathcal{E}|$ identity matrix. Since both $\mathbf{A}$ and $\mathbf{I}$ are determined by the known graph structure, we can simplify Equation (5.7) by defining

$$\mathbf{B} = \mathbf{I} - \mathbf{A}^T(\mathbf{AA}^T)^{-1}\mathbf{A}. \tag{5.8}$$

Then we have

$$\boldsymbol{\Delta}^F = \mathbf{B}\boldsymbol{\Delta}^P. \tag{5.9}$$

It can be easily checked that the final time discrepancy in Figure 5.1.b satisfies (5.9).

### 5.2.2 Communication model.

As proposed by HUYGENS, the calculation of Equation (5.9) is run on a master [40]. Assume each edge information is known by one or both incident server nodes, and each server node can linearly code its known edge information. We aim to minimize the *communication cost* ($CC$), defined as as the number of symbols communicated from all the server nodes to the master node, such that (5.9) can be calculated.

**Trivial scheme.** We can see that HUYGENS algorithm relies on $\mathbf{\Delta}^P$ which is gathered from every pair of neighboring nodes. To obtain $\mathbf{\Delta}^F$, a trivial solution is for each edge, one of the incident nodes should send the information to the master. In this scenario, the communication cost is $CC = |\mathcal{E}|$ symbols.

To motivate the general communication scheme, let us consider the matrix $\mathbf{B}$ in Figure 5.1's example,

$$\mathbf{B} = \begin{bmatrix} 0.5 & -0.25 & -0.25 & -0.25 & -0.25 \\ -0.25 & 0.625 & -0.375 & 0.125 & 0.125 \\ -0.25 & -0.375 & 0.625 & 0.125 & 0.125 \\ -0.25 & 0.125 & 0.125 & 0.625 & -0.375 \\ -0.25 & 0.125 & 0.125 & -0.375 & 0.625 \end{bmatrix}. \tag{5.10}$$

In this case, $rank(\mathbf{B}) = 3$. It means that the potential optimal communication cost is 3 symbols. If every clock knows all the time discrepancies, it is obvious that the communication cost $CC = 3$ because one node can calculate 3 (row) bases of $\mathbf{B}$, denoted by $\mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_3$, and send to the master $\mathbf{B}_1\mathbf{\Delta}^P, \mathbf{B}_2\mathbf{\Delta}^P, \mathbf{B}_3\mathbf{\Delta}^P$. The master can obtain $\mathbf{\Delta}^F = \mathbf{B}\mathbf{\Delta}^P$ from $\mathbf{B}_1\mathbf{\Delta}^P, \mathbf{B}_2\mathbf{\Delta}^P, \mathbf{B}_3\mathbf{\Delta}^P$.

However, each node only has the information of edges that are connected with it. Then, is it possible to achieve the potential optimal communication cost of $rank(\mathbf{B})$? We will answer this question *affirmatively* if the edge information is known by both of the incident nodes.

Next, we describe an equivalent matrix representation of any linear communication scheme, and define the corresponding communication cost. For any graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consider a matrix

$$\mathbf{X}' \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{E}|}, \tag{5.11}$$

whose rows are indexed by the nodes and columns are indexed by the edges. The entry in Row $U \in \mathcal{V}$ and Column $e \in \mathcal{E}$ equals 0 if Node $U$ does not have the time discrepancy information of Edge $e$. Otherwise, the entry can be set as any value. Denote by $\mathbf{x}_U$ the row corresponding to Node $U$. If Node $U$ transmits a symbol to the master, it must be in the form of $\mathbf{x}_U\mathbf{\Delta}^P$. The overall transmitted symbols must be in the form of $\mathbf{X}\mathbf{\Delta}^P$, where rows of $\mathbf{X}$ are chosen from rows of $\mathbf{X}'$ (a row may be chosen multiple times but the non-zero entries can be set differently). Finally, the master obtains $\mathbf{B}\mathbf{\Delta}^P = \mathbf{M}\mathbf{X}\mathbf{\Delta}^P$ for some transformation matrix $\mathbf{M}$. The *communication scheme* consists of the matrices $\mathbf{X}$ and $\mathbf{M}$, such that $\mathbf{B} = \mathbf{M}\mathbf{X}$. The *communication cost* equals the number of rows in $\mathbf{X}$.

## 5.3 Communication-efficient Clock Synchronization

In this section, we show our main results for the communication cost under three cases: the edge information is known by 1) both incident nodes, 2) one incident node, and 3) either one or two incident nodes.

For the first case, we claim that the minimum potential communication cost $rank(\mathbf{B})$ is achievable.

**Theorem 5.1.** For any connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, if every node has all its edges information, there exists an optimal solution where all but one node sends one symbol, and the desired $\mathbf{B}\mathbf{\Delta}^P$ is recovered at the master. The total communication cost $CC = rank(\mathbf{B}) = |\mathcal{V}| - 1$.

For our purpose, in the matrix $\mathbf{X}'$ described in (5.11), row

$$\mathbf{x}_U, U \in \mathcal{V}, \tag{5.12}$$

is defined such that each column (edge) starting from $U$ has entry 1, each column going to $U$ has entry $-1$, and other columns are 0.

We first demonstrate Theorem .1 using the example in Figure 5.1. It can be checked that the following communication scheme satisfies $\mathbf{B} = \mathbf{MX}$:

$$\mathbf{M} = \begin{bmatrix} 0.25 & -0.25 & 0 \\ -0.125 & 0.125 & -0.5 \\ -0.125 & 0.125 & 0.5 \\ 0.375 & 0.625 & 0.5 \\ -0.625 & -0.375 & -0.5 \end{bmatrix}, \tag{5.13}$$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_A \\ \mathbf{x}_B \\ \mathbf{x}_C \end{bmatrix} = \begin{array}{c} \begin{array}{ccccc} AB & BC & CA & BD & DA \end{array} \\ \begin{pmatrix} 1 & 0 & -1 & 0 & -1 \\ -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 \end{pmatrix} \end{array}. \tag{5.14}$$

The transmitted symbols are

$$\mathbf{X}\boldsymbol{\Delta}^P = \begin{bmatrix} l_{AB} - l_{CA} - l_{DA} \\ -l_{AB} + l_{BC} + l_{BD} \\ -l_{BC} + l_{CA} \end{bmatrix} = \begin{bmatrix} 30 \\ -10 \\ 20 \end{bmatrix}. \tag{5.15}$$

This means that we let Nodes $A, B, C$ send a linear combination of the time discrepancies of all their neighbouring edges. In total, the communication cost is $3 = |\mathcal{V}| - 1$ symbols which equals to the potential optimal communication cost $rank(\mathbf{B})$. After the master receives these symbols, it can get $\mathbf{\Delta}^F = \mathbf{M}\mathbf{X}\mathbf{\Delta}^P = [10, -15, 5, 15, -25]^T$.

In fact, for $[\mathbf{x}_\mathrm{A}{}^T, \mathbf{x}_\mathrm{B}{}^T, \mathbf{x}_\mathrm{D}{}^T]^T$, $[\mathbf{x}_\mathrm{A}{}^T, \mathbf{x}_\mathrm{C}{}^T, \mathbf{x}_\mathrm{D}{}^T]^T$, $[\mathbf{x}_\mathrm{B}{}^T, \mathbf{x}_\mathrm{C}{}^T, \mathbf{x}_\mathrm{D}{}^T]^T$, we can also find the corresponding $\mathbf{M}$. In general, it is sufficient to have any $|\mathcal{V}| - 1$ nodes send coded information to the master. Therefore, this scheme tolerances 1 straggler.

The proof of Theorem .1 is broken into several steps. We first show a rank condition in Lemma 1. Then we show the achievability in Lemma 2 and the converse in Lemma 3.

**Lemma 1.** Let $\mathbf{E} \in \mathbb{C}^{m\times n}, \mathbf{F} \in \mathbb{C}^{n\times p}$ be two matrices such that $\mathbf{EF} = \mathbf{0}$, and the null space of $\mathbf{E}$ lies in the column span of $\mathbf{F}$. Then

$$rank(\mathbf{E}) + rank(\mathbf{F}) = n. \tag{5.16}$$

*Proof.* Since null space of $\mathbf{E}$ is in the column span of $\mathbf{F}$,

$$n - rank(\mathbf{E}) \leq rank(\mathbf{F}). \tag{5.17}$$

On the other hand, the rank of the product of $\mathbf{E}$, $\mathbf{F}$ satisfies Sylvester inequality:

$$0 = rank(\mathbf{EF}) \geq rank(\mathbf{E}) + rank(\mathbf{F}) - n. \tag{5.18}$$

Combining (5.17) and (5.18) we see the lemma is proved. □

Define a *loop set* to be a set of loops. We allow disjoint loops as well as loops with common edges. Define the corresponding *loop vector* $\mathbf{y}$ of length $|\mathcal{E}|$ to be a column vector that lists the number of times (with signs) each edge appears in the loop set, which are called the *weights* of the edges. The negative sign means that the edge is in reverse direction. For example, in Figure 5.1, $\{A \to B \to C \to A, A \to B \to D \to A\}$ is a loop set. The loop vector is

$$\begin{array}{ccccc} AB & BC & CA & BD & DA \\ (2 & 1 & 1 & 1 & 1)^T. \end{array} \tag{5.19}$$

The weight of $AB$ is 2, and the weight of $BC$ is 1, etc. For each loop vector, there can be multiple associated loop sets. By the definition of the loop-composition matrix $\mathbf{A}$, a loop vector is a vector in the column span of $\mathbf{A}^T$.

**Lemma 2.** Let $\mathbf{X}$ be the matrix of size $(|\mathcal{V}|-1) \times |\mathcal{E}|$, and its rows be any $|\mathcal{V}|-1$ rows from (5.12). Then, $\mathbf{B} = \mathbf{M}\mathbf{X}$ for $\mathbf{M} = \mathbf{B}\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}$.

*Proof.* We first show $\mathbf{B}^T$ is in the column span of $\mathbf{X}^T$, and hence $\mathbf{B} = \mathbf{M}\mathbf{X}$ for

$$\mathbf{M} = \mathbf{B}\mathbf{X}^{-1}_{right}, \tag{5.20}$$

where $\mathbf{X}^{-1}_{right}$ is the right preudo-inverse of $\mathbf{X}$. Then we find the formula for $\mathbf{M}$.

Assume $\mathbf{a}$ is any column of $\mathbf{A}^T$, which corresponds to a loop. Let $\mathbf{x}_U$ be the row vector as in (5.12) for Node $U$, whose $\pm 1$ entries corresponds to all incident edges of Node $U$. If the loop does not pass Node $U$, then there is no overlap edges in $\mathbf{x}_U$ and $\mathbf{a}$. In this case, $\mathbf{x}_U\mathbf{a} = 0$. Otherwise, every time the loop passes Node $U$, exactly one edge goes into node $U$, and exactly one edge goes out from node $U$. In this case, we also have $\mathbf{x}_U\mathbf{a} = 0$. Therefore, $\mathbf{X}\mathbf{A}^T = \mathbf{0}$.

Now let us prove rows of $\mathbf{X}$ are linearly independent. Consider a vector $\mathbf{y}$ in the null space of $\mathbf{X}$, i.e., $\mathbf{X}\mathbf{y} = 0$. We show $\mathbf{y}$ is a loop vector. Since $\mathbf{x}_U\mathbf{y} = 0$, for any Node $U$, the sum weight in $\mathbf{y}$ for Node $U$'s incoming edges equals the sum weight of its outgoing edges. By Veblen's theorem [9], a directed graph admits a decomposition into directed cycles if and only if the sum weight of the incoming edges equals the sum weight of the outgoing edges for every node. Therefore, $\mathbf{y}$ must be a loop vector, which is in the column span of $\mathbf{A}^T$. Combining the fact that $\mathbf{X}\mathbf{A}^T = \mathbf{0}$, we use Lemma 1 and (5.2) to conclude that

$$rank(\mathbf{X}) = |\mathcal{E}| - rank(\mathbf{A}) = |\mathcal{V}| - 1, \tag{5.21}$$

which equals to the number of rows in $\mathbf{X}$.

Since matrix $\mathbf{A}$ of size $(|\mathcal{E}| - |\mathcal{V}| + 1) \times |\mathcal{E}|$ is full row rank, its null space has dimension $|\mathcal{V}| - 1$. Due to $\mathbf{A}\mathbf{X}^T = \mathbf{0}$ and (5.21), we known the null space of $\mathbf{A}$ equals the column span of $\mathbf{X}^T$. Moreover, consider $\mathbf{B}$ as defined in (5.8),

$$\mathbf{B}\mathbf{A}^T = \mathbf{A}^T - \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{A}^T = \mathbf{A}^T - \mathbf{A}^T = \mathbf{0}. \tag{5.22}$$

Therefore, $\mathbf{B}^T$ is in the null space of $\mathbf{A}$, and hence is in the column span of $\mathbf{X}^T$. Thus, Equation (5.20) holds.

Finally, since $\mathbf{X}$ is full rank, we apply the pseudo-inverse formula to get

$$\begin{aligned} \mathbf{M} &= \mathbf{B}\mathbf{X}^{-1}_{right} && (5.23)\\ &= (\mathbf{I} - \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A})\mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1} && (5.24)\\ &= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}. && (5.25) \end{aligned}$$

In the last step, we used $\mathbf{A}\mathbf{X}^T = \mathbf{0}$. □

**Lemma 3.** $rank(\mathbf{B}) = |\mathcal{V}| - 1$.

*Proof.* We will show the null space of $\mathbf{B}^T$ is in the column span of $\mathbf{A^T}$, and $\mathbf{B}^T\mathbf{A}^T = \mathbf{0}$ so as to use Lemma 1. In that case,

$$rank(\mathbf{B}) = |\mathcal{E}| - rank(\mathbf{A}) = |\mathcal{V}| - 1. \tag{5.26}$$

First,

$$\mathbf{AB} = \mathbf{A} - \mathbf{AA}^T(\mathbf{AA}^T)^{-1}\mathbf{A} = \mathbf{A} - \mathbf{A} = \mathbf{0}, \tag{5.27}$$

and thus $\mathbf{B}^T\mathbf{A}^T = \mathbf{0}$. Second, let $\mathbf{y}$ be any vector in the null space of $\mathbf{B}^T$, namely, $\mathbf{y}^T\mathbf{B} = \mathbf{0}$. Then

$$\begin{aligned} \mathbf{y}^T &= \mathbf{y}^T\mathbf{I} = \mathbf{y}^T\left(\mathbf{B} + \mathbf{A}^T(\mathbf{AA}^T)^{-1}\mathbf{A}\right) \\ &= \left(\mathbf{y}^T\mathbf{A}^T(\mathbf{AA}^T)^{-1}\right) \cdot \mathbf{A}, \end{aligned} \tag{5.28}$$

which belongs to the row span of $\mathbf{A}$. Namely, $\mathbf{y}$ is in the column span of $\mathbf{A}^T$. □

*Proof of Theorem .1.* We can see that the scheme in Lemma 2 has a communication cost of $CC = rank(\mathbf{X}) = |\mathcal{V}| - 1$, matching the minimum potential cost of $rank(\mathbf{B})$ in Lemma 3. Therefore, Theorem .1 is proved. □

We notice that in our optimal scheme, only $|\mathcal{V}| - 1$ nodes transmits one symbol, and we can tolerate 1 straggler.

**Lemma 4.** Let $\mathbf{X}'$ contain all $|\mathcal{V}|$ rows as in (5.12). If the master obtains $\mathbf{X}'\mathbf{\Delta}^P$, then it can perform row operations to get $\mathbf{B\Delta}^P$. Moreover, $rank(\mathbf{X}') = rank(\mathbf{B}) = |\mathcal{V}| - 1$.

*Proof.* From Lemma 2, it is obvious that $\mathbf{X}'$, which includes all rows of $\mathbf{X}$, can be transformed into $\mathbf{B}$ by row operations.

By the same argument as Equation (5.21), one can show that $\mathbf{X}'\mathbf{A}^T = \mathbf{0}$, and the null space of $\mathbf{X}'$ is in the column span of $\mathbf{A}^T$. By Lemma 1 and Lemma 3

$$rank(\mathbf{X}') = |\mathcal{E}| - rank(\mathbf{A}) = |\mathcal{V}| - 1 = rank(\mathbf{B}). \tag{5.29}$$

$\square$

In general, notice that $\mathbf{X}'$ is the incidence matrix of graph $\mathcal{G}$, its rank is

$$rank(\mathbf{X}') = |\mathcal{V}| - n, \tag{5.30}$$

where $n$ is the number of connected components of $\mathcal{G}$ [6, Prop. 4.3].

The above lemma indicates that the master's desired information $\mathbf{B}\mathbf{\Delta}^P$ can be equivalently represented by $\mathbf{X}'\mathbf{\Delta}^P$. From now on, the *desired dimensions* refers to either $\mathbf{B}$ or $\mathbf{X}'$ interchangeably.

Next, we consider the cases where the edge information is known by one or both of the incident nodes. We will state in Lemma 5 a converse of the communication cost, whose proof directly follows from the min-cut bound of linear network computing [2].

To that end, define $\mathcal{E}(U)$ as the set of the edges which are in $\mathcal{E}$ and known by Node $U$, for $U \in \mathcal{V}$. For a set of nodes $\mathcal{U} \subseteq \mathcal{V}$, define $\mathcal{E}(\mathcal{U})$ as the set of the edges known by Nodes $\mathcal{U}$. Denote matrix $\mathbf{X}'_{\mathcal{E}(U)}$ as the sub-matrix of $\mathbf{X}'$ obtained by choosing the columns corresponding to the edges in $\mathcal{E}(U)$. It represents the master's desired dimensions restricted to information known by $U$. Sub-matrix $\mathbf{X}'_{\mathcal{E}(\mathcal{U})}$ is similarly defined.

For example, in Figure 5.1,

$$\mathbf{X}' = \begin{bmatrix} \mathbf{x}_A \\ \mathbf{x}_B \\ \mathbf{x}_C \\ \mathbf{x}_D \end{bmatrix} = \begin{array}{c} \begin{array}{ccccc} AB & BC & CA & BD & DA \end{array} \\ \begin{pmatrix} 1 & 0 & -1 & 0 & -1 \\ -1 & 1 & 0 & 1 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \end{array}. \tag{5.31}$$

Assume the edge information symbols $l_{AB}, l_{CA}, l_{DA}$ are known by Node $A$. Then the desired dimensions known by node $A$ is

$$\mathbf{X}'_{\mathcal{E}(A)} = \begin{array}{c} \begin{array}{ccc} AB & CA & DA \end{array} \\ \begin{pmatrix} 1 & -1 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}, \tag{5.32}$$

which contains a diagonal matrix (ignoring the first row) and has full rank 3. It can be easily seen that for any Node $U \in \mathcal{V}$, $\mathbf{X}'_{\mathcal{E}(U)}$ always contains a diagonal matrix and is full rank:

$$rank(\mathbf{X}'_{\mathcal{E}(U)}) = |\mathcal{E}(U)|. \tag{5.33}$$

**Lemma 5** (Min-cut bound.)**.** Denote $CC_U$ the communication cost from Server $U$. The total communication cost satisfies

$$CC \geq \min \sum_{U \in \mathcal{V}} CC_U, \tag{5.34}$$

$$\text{s.t.} \sum_{U \in \mathcal{U}} CC_U \geq rank(\mathbf{X}'_{\mathcal{E}(\mathcal{U})}), \text{ for all } \mathcal{U} \subseteq \mathcal{V}. \tag{5.35}$$

For example, if we choose $\mathcal{U} = \mathcal{V}$ in (5.35), we get the aforementioned communication cost bound $CC \geq \min \sum_{U \in \mathcal{V}} CC_U \geq rank(\mathbf{X}') = rank(\mathbf{B})$.

The following theorem states that when each edge information is known by only one node, the trivial scheme is optimal.

**Theorem 5.2.** For graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, if each edge information is only known by one of its incident nodes, the optimal solution for the master to obtain the desired $\mathbf{B}\boldsymbol{\Delta}^P$ is to send all the edge information individually. The total communication cost is $CC = |\mathcal{E}|$.

*Proof.* The achievability is obvious. We only need to show the converse, i.e., $CC \geq |\mathcal{E}|$.

The communication cost of any Node $U$ must satisfy

$$CC_U \geq rank\left(\mathbf{X}'_{\mathcal{E}(U)}\right) = |\mathcal{E}(U)|, \tag{5.36}$$

based on the min-cut bound with $\mathcal{U} = \{U\}$ in (5.35) and Equation (5.33). The total communication cost

$$\begin{align} CC &\geq \min \sum_{U \in \mathcal{V}} CC_U \tag{5.37}\\ &\geq \sum_{U \in \mathcal{V}} rank\left(\mathbf{X}'_{\mathcal{E}(U)}\right) \tag{5.38}\\ &= \sum_{U \in \mathcal{V}} |\mathcal{E}(U)| \tag{5.39}\\ &= |\mathcal{E}|, \tag{5.40} \end{align}$$

where the last equality holds since each edge is only known by one node. □

Finally, let us consider the case where each edge information is known by either one or both incident nodes. We say the time discrepancy information on an edge is *singleton* if it is known by just one node.

We provide an achievable scheme in Algorithm 5. It trivially sends singletons to the master, and removes the corresponding edges. Call the remaining graph $\mathcal{H}$, Then it solves the remaining desired dimensions on graph $\mathcal{H}$ as in Lemma 2.

**Algorithm 5** Algorithm for graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where some edge information are singletons.

```
1: Send the singletons directly from the corresponding servers. Let $\mathcal{E}_s$ be the corresponding edges.
2: Let $\mathcal{H}$ be the graph $(\mathcal{V}, \mathcal{E}\backslash\mathcal{E}_s)$.
3: Let $n$ be the number of connected components of $\mathcal{H}$.
4: Let $\mathbf{X}'$ be as in (5.12) for graph $\mathcal{H}$. Let $\mathbf{X}$ be $|\mathcal{V}| - n$ rows of $\mathbf{X}'$ such that one row is excluded from $\mathbf{X}'$ for each connected component.
5: Let $\mathbf{M} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T)^{-1}$.
6: Use the communication scheme $\mathbf{X}, \mathbf{M}$.
7: Combine the singletons and $\mathbf{M}\mathbf{X}\boldsymbol{\Delta}^P$ to obtain $\mathbf{B}\boldsymbol{\Delta}^P$.
```

The following lemma is straightforward from the algorithm.

**Lemma 6.** *Let $m$ be the number of singletons. Let $n$ be the number of connected components of $\mathcal{H}$. Algorithm 5 achieves communication cost of $CC = |\mathcal{V}| - n + m$.*

Algorithm 5 is optimal for certain cases. For example, in Figure 5.1, let $\mathcal{E}(A) = \{AB, CA\}$, $\mathcal{E}(B) = \{BD, BC\}$, $\mathcal{E}(C) = \{CA\}$, $\mathcal{E}(D) = \{DA, BD\}$. There are $m = 3$ singletons, i.e., $l_{AB}, l_{DA}, l_{BC}$. After removing the singleton edges, the graph $\mathcal{H}$ consists of all the 4 nodes but only 2 edges $C \to A$ and $B \to D$. Then $\mathcal{H}$ contains $n = 2$ connected components, such that the first component contains Nodes $A, C$, and the second component contains Nodes $B, D$. The communication cost of Algorithm 5 is $CC = |\mathcal{V}| - n + m = 5$. On the other hand, set

$\mathcal{U} = \{A, C\}$ and $\mathcal{U} = \{B, D\}$ in (5.35),

$$\mathbf{X}'_{\mathcal{E}(\{A,C\})} = \begin{array}{c} \begin{array}{cc} AB & CA \end{array} \\ \begin{pmatrix} 1 & -1 \\ -1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \end{array}, \tag{5.41}$$

$$\mathbf{X}'_{\mathcal{E}(\{B,D\})} = \begin{array}{c} \begin{array}{ccc} BC & BD & DA \end{array} \\ \begin{pmatrix} 0 & 0 & -1 \\ 1 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 1 \end{pmatrix} \end{array}. \tag{5.42}$$

We obtain the lower bound

$$\begin{aligned} CC_A + CC_C &\geq rank\left(\mathbf{X}'_{\mathcal{E}(\{A,C\})}\right) = 2, \\ CC_B + CC_D &\geq rank\left(\mathbf{X}'_{\mathcal{E}(\{B,D\})}\right) = 3, \\ CC &\geq \min \sum_{U \in \mathcal{V}} CC_U \geq 5. \end{aligned}$$

Therefore, the algorithm gives the optimal communication cost in the example.

In general, we have the following sufficient condition for Algorithm 5 to be optimal.

**Theorem 5.3.** Algorithm 5 is optimal under the following condition: $\mathcal{H}$ contains $n \geq 2$ connected components, each component has more than 1 nodes, every singleton edge is between different components, and singleton edges known by distinct nodes in one component are not connected.

*Proof.* We show that the cut-set bound matches the communication cost in Lemma 6. Let $\mathcal{V}_i, \mathcal{E}_i$ be the nodes and edges in the $i$-th connected component of $\mathcal{H}$, and $\mathcal{E}'_i$ the singleton edges known by $\mathcal{V}_i$, $1 \leq i \leq n$. Then, the edges known by the $i$-th component is $\mathcal{E}(\mathcal{V}_i) = \mathcal{E}_i \cup \mathcal{E}'_i$. Set $\mathcal{U} = \mathcal{V}_i$ in (5.35),

$$\mathbf{X}'_{\mathcal{E}(\mathcal{V}_i)} = \begin{array}{c c} & \begin{array}{cc} \mathcal{E}_i & \mathcal{E}'_i \end{array} \\ \begin{array}{c} \mathcal{V}_i \\ \mathcal{V}\backslash\mathcal{V}_i \end{array} & \begin{pmatrix} \mathbf{X}_1 & * \\ \mathbf{0} & \mathbf{X}_2 \end{pmatrix} \end{array}, \tag{5.43}$$

where we list rows (nodes) in the $i$-th component on the top. The matrix $*$ is not of interest. Matrix $\mathbf{X}_1$ is the matrix $\mathbf{X}'$ as in (5.12) for the graph $(\mathcal{V}_i, \mathcal{E}_i)$, whose rank is $|\mathcal{V}_i| - 1$. Matrix $\mathbf{X}_2$ corresponds to the singletons known by the $i$-th component. Due to the given condition on the singletons in the theorem, the subgraph induced by edges $\mathcal{E}'_i$ is simply several disconnected star graphs (one center node connected to other nodes), where nodes in the $i$-th component are the centers. Hence, it can be seen that $\mathbf{X}_2$ it is a diagonal block matrix, and every block corresponds to one star. Since $\mathbf{X}_2$ does not include the rows corresponding to the center nodes in $\mathcal{V}_i$, by (5.30) each block is full rank, which equals to the number of vertices in the star minus 1, or the number of edges. Overall, the diagonal block matrix satisfies $rank(\mathbf{X}_2) = |\mathcal{E}'_i|$. Therefore, the cut-set bound gives

$$\sum_{U \in \mathcal{V}_i} CC_U \geq rank(\mathbf{X}'_{\mathcal{E}(\mathcal{V}_i)}) = |\mathcal{V}_i| - 1 + |\mathcal{E}'_i|. \tag{5.44}$$

The communication cost satisfies

$$
\begin{aligned}
CC &\geq \min \sum_{U \in \mathcal{V}} CC_U && (5.45)\\
&= \min \sum_{i=1}^{n} \sum_{U \in \mathcal{V}_i} CC_U && (5.46)\\
&\geq \sum_{i=1}^{n} (|\mathcal{V}_i| - 1 + |\mathcal{E}_i'|) && (5.47)\\
&= |\mathcal{V}| - n + m. && (5.48)
\end{aligned}
$$

□

## 5.4 Discussion

Besides the clock synchronization, the idea of this work may have other applications. For example, given pairwise evaluations by customers for how much more one item is worth than another, our method can give a communication-efficient global evaluation of all values, subject to minimum norm perturbation.

# Chapter 6

# Conclusion

In this dissertation, we focus on the the source coding and channel coding from coding theory in the data storage and network. Specifically, we investigate the data compression problem in security analysis, information error correcting code in DNA storage and the communication efficiency in clock synchronization problem.

## 6.1 Source coding in data compression for security analysis

Causality analysis reconstructs information flow across different files, processes, and hosts to enable effective attack investigation and forensic analysis. However, it also requires a large amount of storage, which impedes its wide adoption by enterprises. Our work shows the concern about storage overhead can be eased by query-friendly compression. Comparing to prior works based on data reduction, our system `SEAL` offers similar or better storage (e.g., 9.81x event reduction and 2.63x database size reduction on $DS_{ind}$) and query efficiency (average query speed is 64% of the uncompressed form) with guarantee of no false positive and negative in casualty queries. We make the first attempt to integrating the techniques in the coding area (like Delta coding and Golumb coding) with a security application. We hope in the future more security applications can be benefited with techniques from the coding community and we will continue such investigation.

## 6.2 LDPC code in DNA storage

We present binary LDPC codes for DNA storage to combat asymmetric sequencing errors. Our decoding algorithms utilize the extrinsic information exchange to accommodate the dependency of the errors of the two bits in a nucleotide symbol. We demonstrate the desirable performance of our codes in terms of bit error rate and decoding speed.

## 6.3 Constrained code in DNA storage

In conclusion, we present constrained codes for nanopore sequencing in order to limit the synchronization errors. Our encoder improves the state splitting algorithm and achieves a relatively small number of states. Our decoding algorithm uses Viterbi algorithm based on the labeled graph of the constrained code, which makes the code capable for error correction. Simulation result shows that the proposed decoder is effective in limiting errors caused by additive Gaussian noise and deletions.

## 6.4 Communication efficiency in clock synchronization of distributed system

We study the problem of clock synchronization in arbitrary networks. We find the optimal schemes for the two important cases where each time discrepancy is known by both adjacent servers and by only one of the adjacent servers. For the former case, our scheme is also straggler (slow or fail server) robust. For the rest of the cases, we propose an algorithm that outperforms the trivial solution.

# Bibliography

[1] Regular density evolution version 0.1.1. `https://github.com/Sixgods/Density-evolution`. Accessed: 2010-09-30.

[2] R. Appuswamy and M. Franceschetti. Computing linear functions by linear coding over networks. *IEEE Transactions on Information Theory*, 60(1):422–431, 2014.

[3] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger. Network coding for computing: Cut-set bounds. *IEEE Transactions on Information Theory*, 57(2):1015–1030, 2011.

[4] R. Appuswamy, M. Franceschetti, N. Karamchandani, and K. Zeger. Linear codes, target function classes, and network computing capacity. *IEEE Transactions on Information Theory*, 59(9):5741–5753, 2013.

[5] M. Bianchini, M. Gori, and F. Scarselli. Inside pagerank. *ACM Transactions on Internet Technology (TOIT)*, 5(1):92–128, 2005.

[6] N. Biggs, N. L. Biggs, and B. Norman. *Algebraic graph theory*. Number 67. Cambridge university press, 1993.

[7] V. C. Black. Threat hunting and incident response for hybrid deployments. `https://www.carbonblack.com/products/edr/`, 2020.

[8] M. Blawat, K. Gaedke, I. Huetter, X.-M. Chen, B. Turczyk, S. Inverso, B. W. Pruitt, and G. M. Church. Forward error correction for DNA data storage. *Procedia Computer Science*, 80:1011–1022, 2016.

[9] J. Bondy and U. Murty. *Graph theory*. Springer, 2008.

[10] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss. A DNA-based archival storage system. *ACM SIGOPS Operating Systems Review*, 50(2):637–649, 2016.

[11] K. D. Bowers, C. Hart, A. Juels, and N. Triandopoulos. Pillarbox: Combating next-generation malware with fast forward-secure logging. In *International Workshop on Recent Advances in Intrusion Detection*, pages 46–67. Springer, 2014.

[12] A. S. Buyukkayhan, A. Oprea, Z. Li, and W. Robertson. Lens on the endpoint: Hunting for malicious software through endpoint data analysis. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 73–97. Springer, 2017.

[13] B. Cantrill, M. W. Shapiro, and A. H. Leventhal. Dynamic Instrumentation of Production Systems. In *{USENIX} Annual Technical Conference (ATC)*, Boston, MA, 2004.

[14] S. Chandak, J. Neu, K. Tatwawadi, J. Mardia, B. Lau, M. Kubit, R. Hulett, P. Griffin, M. Wootters, T. Weissman, et al. Overcoming high nanopore basecaller error rates for dna storage via basecaller-decoder integration and convolutional codes. *bioRxiv*, 2019.

[15] S. Chandak, J. Neu, K. Tatwawadi, J. Mardia, B. Lau, M. Kubit, R. Hulett, P. Griffin, M. Wootters, T. Weissman, et al. Overcoming high nanopore basecaller error rates for dna storage via basecaller-decoder integration and convolutional codes. pages 8822–8826, 2020.

[16] S. Chandak, K. Tatwawadi, B. Lau, J. Mardia, M. Kubit, J. Neu, P. Griffin, M. Wootters, T. Weissman, and H. 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. IEEE, 2019.

[17] Y. M. Chee, H. M. Kiah, and H. Wei. Efficient and explicit balanced primer codes. *IEEE Transactions on Information Theory*, 2020.

[18] Y. M. Chee and S. Ling. Improved lower bounds for constant GC-content DNA codes. *IEEE Transactions on Information Theory*, 54(1):391–394, 2008.

[19] K. Chen, J. Zhu, F. Boskovic, and U. F. Keyser. Nanopore-based DNA hard drives for rewritable and secure data storage. *Nano letters*, 20(5):3754–3760, 2020.

[20] M. Cheraghchi, J. Ribeiro, R. Gabrys, and O. Milenkovic. Coded trace reconstruction. In *2019 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2019.

[21] G. M. Church, Y. Gao, and S. Kosuri. Next-generation digital information storage in DNA. *Science*, page 1226355, 2012.

[22] L. Conde-Canencia and L. Dolecek. Nanopore DNA sequencing channel modeling. In *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, pages 258–262. IEEE, 2018.

[23] G. V. Cormack. Data compression on a database system. *Communications of the ACM*, 28(12):1336–1342, 1985.

[24] Cybereason. EDR — cybereason defense platform. `https://www.cybereason.com/platform/endpoint-detection-response-edr`. Accessed: 2020-2-15.

[25] DARPA/I2O. DARPA Transparent Computing. `https://github.com/darpa-i2o/Transparent-Computing`, 2020.

[26] A. Dasgupta, R. Kumar, and T. Sarlos. On estimating the average degree. In *Proceedings of the 23rd international conference on World wide web*, pages 795–806, 2014.

[27] M. David, L. J. Dursi, D. Yao, P. C. Boutros, and J. T. Simpson. Nanocall: an open source basecaller for oxford nanopore sequencing data. *Bioinformatics*, 33(1):49–55, 2017.

[28] P. Deutsch et al. Gzip file format specification version 4.3. Technical report, RFC 1952, May, 1996.

[29] M. Du, F. Li, G. Zheng, and V. Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, 2017.

[30] Y. Erlich and D. Zielinski. DNA fountain enables a robust and efficient storage architecture. *Science*, 355(6328):950–954, 2017.

[31] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 157–168, 2012.

[32] U. Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.

[33] J. Feng, K. Liu, R. D. Bulushev, S. Khlybov, D. Dumcenco, A. Kis, and A. Radenovic. Identification of single nucleotides in mos 2 nanopores. *Nature nanotechnology*, 10(12):1070, 2015.

[34] FireEye. Endpoint security software and solutions. `https://www.fireeye.com/solutions/hx-endpoint-security-products.html`, 2020.

[35] G. D. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[36] R. Gabrys, H. M. Kiah, and O. Milenkovic. Asymmetric lee distance codes for DNA-based storage. *IEEE Transactions on Information Theory*, 63(8):4982–4995, 2017.

[37] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal. {SAQL}: A stream-based query system for real-time abnormal system behavior detection. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 639–656, 2018.

[38] P. Gao, X. Xiao, Z. Li, K. Jee, F. Xu, S. R. Kulkarni, and P. Mittal. A query system for efficiently investigating complex attack behaviors for enterprise security. *Proceedings of the VLDB Endowment*, 12(12):1802–1805, 2019.

[39] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal. {AIQL}: Enabling efficient attack investigation from system monitoring data. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 113–126, 2018.

[40] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosunblum, and A. Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, NSDI'18, page 81–94. USENIX Association, 2018.

[41] D. G. Gibson, L. Young, R.-Y. Chuang, J. C. Venter, C. A. Hutchison, and H. O. Smith. Enzymatic assembly of DNA molecules up to several hundred kilobases. *Nature methods*, 6(5):343–345, 2009.

[42] A. Giridhar and P. R. Kumar. Distributed clock synchronization over wireless networks: Algorithms and analysis. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 4915–4920. IEEE, 2006.

[43] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77, 2013.

[44] O. Goldreich and D. Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008.

[45] S. W. Golomb, B. Gordon, and L. R. Welch. Comma-free codes. *Canadian Journal of Mathematics*, 10:202–209, 1958.

[46] X. Guang, R. W. Yeung, S. Yang, and C. Li. Improved upper bound on the network function computing capacity. *IEEE Transactions on Information Theory*, 65(6):3790–3811, 2019.

[47] Guppy. Guppy. `https://nanoporetech.com/`.

[48] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer. UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *NDSS*, San Diego, CA.

[49] M. H. Hansen and W. N. Hurwitz. On the theory of sampling from finite populations. *The Annals of Mathematical Statistics*, 14(4):333–362, 1943.

[50] R. Heckel, I. Shomorony, K. Ramchandran, and N. David. Fundamental limits of DNA storage systems. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 3130–3134. IEEE, 2017.

[51] K. J. Higgins. The rebirth of endpoint security. `https://www.darkreading.com/%20endpoint/the-rebirth-of-endpoint-security/d/d-id/1322775`, 2015.

[52] M. N. Hossain, J. Wang, O. Weisse, R. Sekar, D. Genkin, B. He, S. D. Stoller, G. Fang, F. Piessens, E. Downing, et al. Dependence-preserving data compaction for scalable forensic analysis. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1723–1740, 2018.

[53] C. Huang, Z. Tan, S. Yang, and X. Guang. Comments on cut-set bounds on network function computation. *IEEE Transactions on Information Theory*, 64(9):6454–6459, 2018.

[54] K. A. S. Immink and K. Cai. Design of capacity-approaching constrained codes for DNA-based storage systems. *IEEE Communications Letters*, 22(2):224–227, 2017.

[55] M. Jain, H. E. Olsen, B. Paten, and M. Akeson. The oxford nanopore minion: delivery of nanopore sequencing to the genomics community. *Genome biology*, 17(1):239, 2016.

[56] V. Karande, E. Bauman, Z. Lin, and L. Khan. Sgx-log: Securing system logs with sgx. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 19–30, 2017.

[57] L. Katzir, E. Liberty, and O. Somekh. Estimating sizes of social networks via biased sampling. In *Proceedings of the 20th international conference on World wide web*, pages 597–606, 2011.

[58] A. Kavcic, X. Ma, and M. Mitzenmacher. Binary intersymbol interference channels: Gallager codes, density evolution, and code performance bounds. *IEEE Transactions on Information theory*, 49(7):1636–1652, 2003.

[59] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *2011 IEEE 27th International Conference on Data Engineering*, pages 195–206. IEEE, 2011.

[60] S. T. King and P. M. Chen. Backtracking intrusions. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 223–236, 2003.

[61] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Transactions on Computers*, 100(8):933–940, 1987.

[62] K. H. Lee, X. Zhang, and D. Xu. Loggc: garbage collecting audit log. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1005–1016. ACM, 2013.

[63] M. Leng and Y.-C. Wu. Distributed clock synchronization for wireless sensor networks using belief propagation. *IEEE Transactions on Signal Processing*, 59(11):5404–5414, 2011.

[64] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi. Anchor-based correction of substitutions in indexed sets. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 757–761. IEEE, 2019.

[65] H. Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[66] Y. Li, G. Kumar, H. Hariharan, H. Wassel, P. Hochschild, D. Platt, S. Sabato, M. Yu, N. Dukkipati, P. Chandra, et al. Sundial: Fault-tolerant clock synchronization for datacenters. In *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pages 1171–1186, 2020.

[67] Y. Li, S. Wang, C. Bi, Z. Qiu, M. Li, and X. Gao. Deepsimulator1. 5: a more powerful, quicker and lighter simulator for nanopore sequencing. *Bioinformatics*, 36(8):2578–2580, 2020.

[68] D. Limbachiya, M. K. Gupta, and V. Aggarwal. Family of constrained codes for archival DNA data storage. *IEEE Communications Letters*, 22(10):1972–1975, 2018.

[69] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1777–1794, 2019.

[70] R. Lopez, Y.-J. Chen, S. D. Ang, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Seelig, K. Strauss, and L. Ceze. DNA assembly for nanopore data storage readout. *Nature communications*, 10(1):1–9, 2019.

[71] A. Maccioni and D. J. Abadi. Scalable pattern matching over compressed graphs via dedensification. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1755–1764, 2016.

[72] M. K. Maggs, S. G. O'keefe, and D. V. Thiel. Consensus clock synchronization for wireless sensor networks. *IEEE sensors Journal*, 12(6):2269–2277, 2012.

[73] E. Mallada, X. Meng, M. Hack, L. Zhang, and A. Tang. Skewless network clock synchronization without discontinuity: Convergence and performance. *IEEE/ACM Transactions on Networking*, 23(5):1619–1633, 2014.

[74] E. Manzoor, S. M. Milajerdi, and L. Akoglu. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *SIGKDD*, pages 1035–1044, New York, New York, USA, 2016. ACM Press.

[75] W. Mao, S. N. Diggavi, and S. Kannan. Models and information-theoretic bounds for nanopore sequencing. *IEEE Transactions on Information Theory*, 64(4):3216–3236, 2018.

[76] A. Marathe, A. E. Condon, and R. M. Corn. On combinatorial DNA word design. *Journal of Computational Biology*, 8(3):201–219, 2001.

[77] B. H. Marcus, R. M. Roth, and P. H. Siegel. An introduction to coding for constrained systems. *Lecture notes*, 2001.

[78] H. Maserrat and J. Pei. Neighbor query friendly compression of social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 533–542, 2010.

[79] Microsoft. Event tracing for windows (etw). `https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/event-tracing-for-windows--etw-`, 2017.

[80] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan. Holmes: real-time apt detection through correlation of suspicious information flows. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1137–1152. IEEE, 2019.

[81] J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein. Delta encoding in http. *IETF, Gennaio*, 65, 2002.

[82] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[83] H. Nguyen, R. Ivanov, L. T. Phan, O. Sokolsky, J. Weimer, and I. Lee. Logsafe: secure and scalable data logger for iot devices. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 141–152. IEEE, 2018.

[84] T. T. Nguyen, K. Cai, K. A. S. Immink, and H. M. Kiah. Constrained coding with error control for DNA-based data storage. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 694–699. IEEE, 2020.

[85] C. Nill and C.-E. Sundberg. List and soft symbol output viterbi algorithms: Extensions and comparisons. *IEEE Transactions on Communications*, 43(2/3/4):277–287, 1995.

[86] W. S. Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[87] M. OpenCourseWare. Left and right inverses; pseudoinverse. *Online. http://ocw. mit. edu/courses/mathematics/18-06sc-linear-algebra-fall-2011/positive-definite-matrices-and-applications/left-and-right-inverses-pseudoinverse/MIT18_06SCF11_Ses3. 8sum. pdf. Accessed March*, 2015.

[88] A. Oprea, Z. Li, R. Norris, and K. Bowers. Made: Security analytics for enterprise threat detection. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 124–136, 2018.

[89] A. Oprea, Z. Li, T.-F. Yen, S. H. Chin, and S. Alrwais. Detection of early-stage enterprise infection by mining large-scale log data. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 45–56. IEEE, 2015.

[90] L. Organick, S. D. Ang, Y.-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen, et al. Random access in large-scale DNA data storage. *Nature biotechnology*, 36(3):242, 2018.

[91] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. W. Fletcher, A. Miller, and D. Tian. Custos: Practical tamper-evident auditing of operating systems using trusted execution. In *Proc. of the Symposium on Network and Distributed System Security (NDSS)*, 2020.

[92] R. Palanki, A. Khandekar, and R. McEliece. Graph-based codes for synchronous multiple access channels. In *Proceedings of the Annual Allerton Conference on Communication Control and Computing*, volume 39, pages 1263–1271. The University; 1998, 2001.

[93] Y. S. Patel, A. Page, M. Nagdev, A. Choubey, R. Misra, and S. K. Das. On demand clock synchronization for live vm migration in distributed cloud data centers. *Journal of Parallel and Distributed Computing*, 138:15–31, 2020.

[94] R. Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, 1955.

[95] Redhat. Chapter 7. system auditing. `https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/security_guide/chap-system_auditing`, 2019.

[96] T. Richardson and R. Urbanke. *Modern coding theory.* Cambridge university press, 2008.

[97] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE transactions on information theory*, 47(2):619–637, 2001.

[98] W. Ryan and S. Lin. *Channel codes: classical and modern.* Cambridge University Press, 2009.

[99] C. Schoeny, F. Sala, and L. Dolecek. Novel combinatorial coding results for dna sequencing and data storage. In *Signals, Systems, and Computers, 2017 51st Asilomar Conference on*, pages 511–515. IEEE, 2017.

[100] N. Seshadri and C.-E. Sundberg. List viterbi decoding algorithms with applications. *IEEE transactions on communications*, 42(234):313–323, 1994.

[101] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[102] C. Shepherd, R. N. Akram, and K. Markantonakis. Emlog: tamper-resistant system logging for constrained devices with tees. In *IFIP International Conference on Information Security Theory and Practice*, pages 75–92. Springer, 2017.

[103] T. Shinkar, E. Yaakobi, A. Lenz, and A. Wachter-Zeh. Clustering-correcting codes. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 81–85. IEEE, 2019.

[104] I. Shomorony and R. Heckel. Capacity results for the noisy shuffling channel. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 762–766. IEEE, 2019.

[105] J. Sima, N. Raviv, and J. Bruck. On coding over sliced information. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 767–771. IEEE, 2019.

[106] I. Smagloy, L. Welter, A. Wachter-Zeh, and E. Yaakobi. Single-deletion single-substitution correcting codes. In *2020 IEEE International Symposium on Information Theory (ISIT)*, pages 775–780. IEEE, 2020.

[107] R. Solis, V. S. Borkar, and P. R. Kumar. A new distributed time synchronization protocol for multihop wireless networks. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 2734–2739. IEEE, 2006.

[108] W. Song, K. Cai, and K. A. S. Immink. Sequence-subset distance and coding for error control for dna-based data storage. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 86–90. IEEE, 2019.

[109] W. Song, K. Cai, M. Zhang, and C. Yuen. Codes with run-length and GC-content constraints for DNA-based data storage. *IEEE Communications Letters*, 22(10):2004–2007, 2018.

[110] K. Suyehira, S. Llewellyn, R. M. Zadegan, W. L. Hughes, and T. Andersen. A coding scheme for nucleic acid memory (nam). In *2017 IEEE Workshop on Microelectronics and Electron Devices (WMED)*, pages 1–3. IEEE, 2017.

[111] Y. Tang, D. Li, Z. Li, M. Zhang, K. Jee, X. Xiao, Z. Wu, J. Rhee, F. Xu, and Q. Li. Nodemerge: template based efficient data reduction for big-data causality analysis. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1324–1337. ACM, 2018.

[112] H. Tavakoli, M. A. Attari, and M. R. Peyghami. Optimal rate for irregular ldpc codes in binary erasure channel. In *2011 IEEE Information Theory Workshop*, pages 125–129. IEEE, 2011.

[113] O. N. Technologies. kmer models from Oxford Nanopore Technologies. `https://github.com/nanoporetech/kmer_models`, 2016.

[114] G. Tenengolts. Nonbinary codes, correcting single deletion or insertion (corresp.). *IEEE Transactions on Information Theory*, 30(5):766–769, 1984.

[115] W. Timp, J. Comer, and A. Aksimentiev. DNA base-calling from a nanopore using a Viterbi algorithm. *Biophysical journal*, 102(10):L37–L39, 2012.

[116] Trustwave. Trustwave global security report, 2015.

[117] J. Uthayakumar, T. Vengattaraman, and P. Dhavachelvan. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *Journal of King Saud University-Computer and Information Sciences*, 2018.

[118] N. Van Tu, J. Hyun, G. Y. Kim, J.-H. Yoo, and J. W.-K. Hong. Intcollector: A high-performance collector for in-band network telemetry. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 10–18. IEEE, 2018.

[119] A. Viterbi. Convolutional codes and their performance in communication systems. *IEEE Transactions on Communication Technology*, 19(5):751–772, 1971.

[120] Y. Wang, M. Noor-A-Rahim, E. Gunawan, Y. L. Guan, and C. L. Poh. Construction of bio-constrained code for DNA data storage. *IEEE Communications Letters*, 23(6):963–966, 2019.

[121] A. C. Ward and W. Kim. Minion™: New, long read, portable nucleic acid sequencing device. *Journal of Bacteriology and Virology*, 45(4):285–303, 2015.

[122] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang. High fidelity data reduction for big data security dependency analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 504–516. ACM, 2016.

[123] S. H. T. Yazdi, R. Gabrys, and O. Milenkovic. Portable and error-free DNA-based data storage. *Scientific reports*, 7(1):5011, 2017.

[124] S. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic. A rewritable, random-access DNA-based storage system. *Scientific reports*, 5:14138, 2015.

[125] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *Proceedings of the 29th Annual Computer Security Applications Conference*, pages 199–208, 2013.

[126] J. Zhan, S. Y. Park, M. Gastpar, and A. Sahai. Linear function computation in networks: Duality and constant gap results. *IEEE Journal on Selected Areas in Communications*, 31(4):620–638, 2013.

# Supplemental Material

## Compression Ratio as a function of Average Degrees

In this section, we derive explicit expressions for the compression ratio. We show that the compressed graph size is always smaller than the original size, though new vertices might be introduced.

In a dependency graph $G = (V, E)$, the number of vertices (nodes) is denoted by $n = |V|$, and the number of edges is denoted by $m = |E|$. Recall that the edges are directed and multiple edges (repeated edges) may exist from one node to another. For node $v \in V$, let its number of parent nodes be $p_v$, and its number of incoming edges be $m_v$. We have $m = \sum_{v \in V} m_v$. Moreover, denote by $p = \sum_{v \in V} p_v$ the total number of parent nodes for all nodes in $V$. Therefore, $p$ represents the number of edges of $G$ after removing repeated ones. Let $G_{undirected}$ denote the undirected graph which is identical to $G$ except that edge directions are removed. Let $G_{simple}$ denote the simple graph obtained by removing the edge directions and the repeated edges from $G$. The average degree of the graph $G_{undirected}$ is denoted by $d_{avg}$. Then,

$$d_{avg} = \frac{2m}{n} = \frac{2 \sum_{v \in V} m_v}{n}. \tag{1}$$

The average degree of $G_{simple}$ is denoted by $p_{avg}$, which is

$$p_{avg} = \frac{2p}{n} = \frac{2 \sum_{v \in V} p_v}{n}. \tag{2}$$

Denote by $S_{event}$, $S_{node}$ the event and node sizes before compression, and by $S'_{event}$, $S'_{node}$ the sizes after compression. They can be calculated by

$$S_{event} = \sum_{v \in V} m_v C_{event}, \tag{3}$$

$$S'_{event} = \sum_{v \in V : m_v > 1} \Big( C_{event} + 2 m_v C_\Delta \Big) + \sum_{v \in V : m_v = 1} C_{event}, \tag{4}$$

$$S_{node} = n C_{node}, \tag{5}$$

$$S'_{node} = n C_{node} + \mathit{size\_map}. \tag{6}$$

Here $C_{event} = 105$ (measured in bytes) is the size of all attributes of an event in the uncompressed format. In our database, $C_{event}$ includes the sizes of `starttime` , `endtime`, `agentid`, etc. $C_\Delta$ is the delta-encoded data and separator size for each time entry, and the factor 2 reflects that two time attributes are recorded for every event. For most of the cases we have observed, $C_\Delta \leq 4$ bytes. $C_{node}$ is the size of one node entry in the uncompressed format, including the size of `nodeid, nodename`, etc. Finally, *size_map* is the node map shown in table 2.2, and can be expressed as

$$\mathit{size\_map} = \sum_{v \in V} C_{ID}(p_v + 1). \tag{7}$$

Here $C_{ID} = 4$ is the constant size required for each `nodeid`. The above size parameters depend on the particular database attributes, and to allow for an arbitrary database design, we use the general expressions instead of the particular sizes. In our experiments, $S_{node}$ and $S'_{node}$ take a negligible fraction of the total storage. As a result, we ignore the node sizes in the following calculations. However, an exact calculation can be carried out if the node size is comparable to the event size.

The difference between the original size and the compressed size is:

$$S_{event} - S'_{event} \tag{8}$$

$$= \sum_{v \in V: m_v > 1} \Big( m_v (C_{event} - 2C_{\Delta}) - C_{event} \Big). \tag{9}$$

It is obvious that the compressed size is always smaller than the original size if $C_{event} > 2C_{\Delta}$, which is true in our deployment. The compression ratio can be expressed as

$$ratio = \frac{S_{event}}{S'_{event}} \tag{10}$$

$$\geq \frac{\sum_{v \in V} m_v C_{event}}{\sum_{v \in V} \Big( C_{event} + 2 m_v C_{\Delta} \Big)} \tag{11}$$

$$= \frac{m C_{event}}{n C_{event} + 2 m C_{\Delta}} \tag{12}$$

$$= \frac{d_{avg} C_{event}}{2 C_{event} + 2 d_{avg} C_{\Delta}} \tag{13}$$

Equation (11) holds because we remove the condition $m_v > 1$ in the denominator, and thus we obtain a lower bound on the ratio. In Equation (13) we multiply the numerator and the denominator by $\frac{2}{n}$ and used Equation (1). If the node size is also included, the ratio will also depend on $p_{avg}$ defined in Equation (2).

**Remark.** 1) Let us call the dependency graph "incompressible" if its compression ratio is lower than a given threshold. It is often unacceptable to compress graphs that are incompressible. Therefore, our estimated compression ratio is a lower bound of the exact ratio (e.g., the inequality of Equation (11)). 2) The discussion in Section 2.3.6 assumes that the node map size is negligible. If it is not, we also need to estimate $p_{avg}$. Note that $p_{avg}$ corresponds to the average degree of the simple graph $G_{simple}$, one can simply apply Algorithm 2 to $H = G_{simple}$.

## Average Degree Estimator Evaluation

We measure the performance of the compression ratio (or average degree) estimator on our dataset following Algorithm 2. We run the algorithm on 8 chunks, each containing $10^6$ events. For each chunk, 20 independent trials are conducted. The parameters are chosen to be $\theta = 10, p_{jump} = 0.1$, such that the estimation error is minimized for the chunks in the experiment. We measure the mean squared error (MSE) between the estimated average degree $\hat{d}$ and the true average degree $d_{avg}$, averaged over all trials and all chunks in the experiment. The results are shown in Figure 1. The MSE value has an obvious drop as the sample size percentage grows up to 5% and quickly converges when the samples cover half of the whole trunk. We then set 5% as the sample size.

Figure 1 also shows that our method has better accuracy compared to the naive estimator, which estimates $d_{avg}$ by directly calculating the average degrees of uniformly sampled nodes.

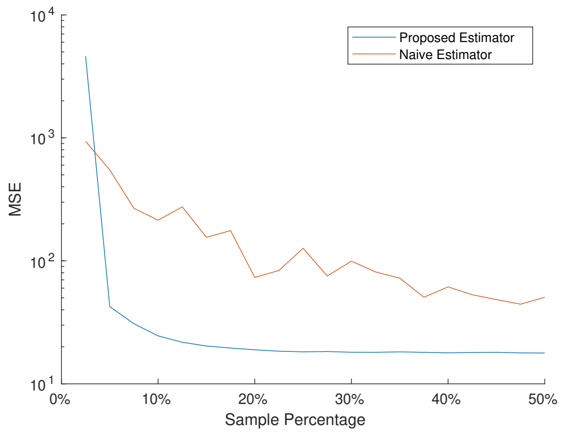


Figure 1: The y axis is the mean square error distance between the estimated average degree from the sampled data and the true average degree, averaged over all trials and all chunks in the experiment. The x axis is the percentage of the sample.

## A Simple Model for the Synchronization Error Rate

While the synchronization error probability is different among base-calling algorithms, we assume the following simple model to capture the fact that the synchronization error rate increases with the similarity of adjacent current signals:

$$\Pr(y_i \text{ syn err}|y_{i-1}, y_i) = \frac{1}{1 + e^{-k(|y_i - y_{i-1}| - d_0)}}. \tag{14}$$

Here $k_0 > 0, d_0 \geq 0$ are parameters dependent on the base-caller, controlling the steepness and the midpoint of the curve, respectively. The overall synchronization error rate can be computed as

$$\begin{aligned} \Pr(\text{syn err}) = \sum_{(x_{i-1},\dots,x_{i+5})} \Big( & P(x_{i-1}, \dots, x_{i+5}) \\ \times \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} & f(y_{i-1}|x_{i-1}, \dots, x_{i+4}) f(y_i|x_i, \dots, x_{i+5}) \\ \times & \Pr(y_i \text{ syn err}|y_{i-1}, y_i) \mathrm{d}y_{i-1} \mathrm{d}y_i. \Big) \end{aligned} \tag{15}$$

Here $P(x_{i-1}, \dots, x_{i+5})$ is the probability the 7-mer occurs in a DNA codeword, which is $\frac{1}{4^7}$ when there are no constraints and all nucleotide sequences are equally likely. And $f(\cdot|\cdot)$ is the probability density function of the Gaussian random variable.

We show the relation of synchronization error rate Pr(syn err) in (15) and the capacity $cap(\mathcal{C})$ in (4.3) with different $t$ in Table 1. For a fixed $t$, our code is constructed from the encoder described in Section 4.2.3. When applying Eq. (15), the term $P(x_{i-1}, \dots, x_{i+5})$ is computed as the steady-state probability of the 7-mer assuming all the outgoing edges of one state has equal transition probability. We set $d_0 = 0$ in Eq. (14) to cover the most effective base-callers. We also notice that different $k_0$ has minor effect for Pr(syn err), so we only show the result when $k_0 = 1$. It can be seen from the table that one can reduce the synchronization error by one quarter at the expense of 12.7% capacity, when $t$ is increased from 1 to 5.

Table 1: The relation between the synchronization error rate and the capacity for different threshold $t$.

|  | $t = 1$ | $t = 3$ | $t = 5$ | $t = 8$ | $t = 10$ |
|---|---|---|---|---|---|
| syn error rate | 0.02502 | 0.02561 | 0.01872 | 0.01563 | 0.01063 |
| symbol capacity | 0.9726 | 0.9166 | 0.8494 | 0.7391 | 0.6338 |

## Proof of Theorem 1

*Proof.* We prove the statement by induction on the row index.

**Base case:** It is easy to see that for $j = 0$ or $j = i$, $sc(i, j) = sc'(i, j)\epsilon^{i-j}$.

**Induction step:** assume for rows $i - 1$, (4.7) holds. Then the score of the $(i, j)$-th entry (for $i > j$) becomes

$$\begin{aligned} sc(i,j) &= \max\{sc(i-1,j-1)p_{i,j}, sc(i-1,j)\epsilon\} && (16)\\ &= \max\{sc'(i-1,j-1)p_{i,j}\epsilon^{i-j}, sc'(i-1,j)\epsilon^{i-1-j}\epsilon\} && (17)\\ &= \max\{sc'(i-1,j-1)p_{i,j}, sc'(i-1,j)\}\epsilon^{i-j} && (18)\\ &= sc'(i,j)\epsilon^{i-j}. && (19) \end{aligned}$$

The proof is completed. □ □

## Anticipation

When no deletions are considered, one can employ a direct constrained code decoder: Set the initial state as $u_0$, which is the same as the initial state of the encoder. For Stage $j = 0, 1, \ldots,$ given the $q$ current signals $(y_{jq+1}, y_{jq+2}, \ldots, y_{jq+q})$, select the edge from $u_j$ whose output label has the highest posterior probability, and set $u_{j+1}$ as the corresponding destination state. The direct decoder is optimal for every particular stage, but not for the entire sequence. It is suitable when there are no deletions, and low complexity is required due to, e.g., limited computing resources. However, certain cases cause ambiguity in decoding the information unless the decoder "looks ahead" at several stages, as explained below.

In Figure 4.3.e, state splitting creates a pair of edges with the same output label $CATCG$ for $S_1$. It brings a problem for the direct decoder: we can not know whether the input label is $AAAA$ or $AAAT$ if the current state is $S_1$ and the maximum a posteriori codeword is $CATCG$. In this case, we have to check the next stage from both $S_2$ and $S_2'$ to help identify the correct input label. The number of stages to look ahead in order to determine the input label is directly related to the delay and the complexity of the decoder, which is captured by the notion of anticipation.

The *anticipation* of State $u$ is defined as the smallest integer $a(u)$ such that any two paths starting from $u$ of length $a(u)+1$ and with the same output label must have the same initial edge. In other words, we can determine the initial edge of a path if we know the initial state $u$ and the first $a(u)+1$ output labels that it generates. The anticipation of the graph is the largest anticipation of the states.

In our transforms of graphs in Figure 4.3, the anticipation remains 0 until spitting. In particular, anticipation of a state possibly increases only when the child state is split. We have the following bounds on the anticipation for every splitting step.

**Theorem .1.** Let State $v$ be any of the parents of State $u$. Let $a(u)$ and $a(v)$ be their anticipation before splitting. After splitting State $u$, the new anticipation of $v$ is bounded as

$$a_{new}(v) \leq \max\{a(v), a(u) + 1\}. \tag{20}$$

In order to prove the theorem, we introduce a few notations and formally present the state splitting rule.

**Notation.** Suppose an edge $e$ in the labeled graph is from state $v$ to state $u$, then we write $e = (v, u)$. For some integer index $i$, let $p_i = (p_{i1}, p_{i2}, \ldots, p_{iN})$ be a path connecting edges $p_{i1}, p_{i2}, ..., p_{iN}$.

**State splitting rule:**

- Let $u$ be the state to be split. It is replicated into states $u^1, u^2$.
- Each incoming edge of $u$ excluding self-loops is duplicated. Namely, $e = (v, u)$, $v \neq u$, is duplicated to be $e^1 = (v, u^1)$, $e^2 = (v, u^2)$.
- Let the outgoing edges of $u$ be $E_u$ and partition it as $E_u = E_u^1 \cup E_u^2$. For each outgoing edge of $u$, $e = (u, v)$ is changed to $e^1 = (u^1, v)$ if $e \in E_u^1$, or $e^1 = (u^2, v)$ if $e \in E_u^1$. The superscript of $e$ is 1 because the edge is not duplicated.
- For any remaining edge $e$, it is not changed. Denote by $e^1$ the corresponding edge after splitting.
- The label of $e^1$ (and $e^2$ if it exists) remains $L(e)$, for any edge $e$.

*Proof of Theorem 2.* Let $\mathbf{H}$ be the graph before splitting state $u$, and $\mathbf{H}'$ the graph after one step of splitting.

We prove the theorem by contradiction. Assume the anticipation $a_{new}(v) \geq \max\{a(v), a(u)+1\}+1$. Then by the minimality of anticipation, there exist two paths in $\mathbf{H}'$ from State $v$ of length $N = \max\{a(v), a(u)+1\}+1$, and of identical labels but their initial edges are different. Denote them as

$$p_1' = (p_{11}^{i_{11}}, p_{12}^{i_{12}}, \ldots, p_{1N}^{i_{1N}}),$$

$$p_2' = (p_{21}^{i_{21}}, p_{22}^{i_{22}}, \ldots, p_{2N}^{i_{2N}}),$$

where each edge $p_{\ell,j}^{i_{\ell,j}}$ corresponds to edge $p_{\ell,j}$ in $\mathbf{H}$ and some index $i_{\ell,j} \in \{1,2\}$, $1 \leq \ell \leq 2, 1 \leq j \leq N$.

Consider the following two paths in graph $\mathbf{H}$

$$p_1 = (p_{11}, p_{12}, \ldots, p_{1N}),$$

$$p_2 = (p_{21}, p_{22}, \ldots, p_{2N}).$$

It can be easily seen that the above are indeed paths in $\mathbf{H}$ by the splitting rule. They both start from state $v$ and are of identical labels. We will show they violate the anticipation of state $v$ or state $u$.

Since the initial edges are different, i.e., $p_{11}^{i_{11}} \neq p_{21}^{i_{21}}$, there are two possible cases.

**Case I.** $p_{11} \neq p_{21}$. Thus, paths $p_1, p_2$ have different initial edges, but their length is at least $a(v)+1$, violating the anticipation of $v$.

**Case II.** $p_{11} = p_{21}, i_{11} \neq i_{21}$. Assume without of generality that $i_{11} = 1, i_{21} = 2$. In this case, the edge $p_{11} = p_{21}$ must start from $v$ and end at $u$. Hence, $p_{12}$ and $p_{22}$ must be outgoing edges of $u$ and must belong to different partitions: $p_{12} \in E_u^1, p_{22} \in E_u^2$, and $i_{21}, i_{22}$ must be 1

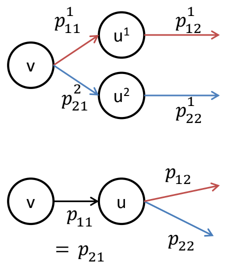


Figure 2: Case II for the proof of Theorem 2. The top graph shows the two paths after splitting. The bottom graph shows the corresponding paths before splitting.

(see Figure 2). Thus $p_{12} \neq p_{22}$. Now consider the paths in $\mathbf{H}$:

$$p_3 = (p_{12}, \ldots, p_{1N}),$$

$$p_4 = (p_{22}, \ldots, p_{2N}).$$

They both start from $u$, have length at least $a(u)+1$, but have different initial edges, which violates the anticipation of $u$. $\square$ $\square$

According to Theorem .1, our greedy splitting ordering rule is as follows. Let $\mathbf{H}$ represent the induced subgraph of $\mathbf{G}_m$ including all states to be split and the edges between them. Choose the state with the least number of parents in $\mathbf{H}$ and split it, and remove that state from $\mathbf{H}$. Repeat until there is no state to split. In our experiment, the maximum anticipation is only 2.