

An error correction algorithm for long-read sequencing

A THESIS
SUBMITTED FOR THE DEGREE OF
Master of Technology (Research)
IN THE
Faculty of Engineering

BY
Parvesh Barak



Department of Computational and Data Sciences
Indian Institute of Science
Bangalore – 560 012 (INDIA)

September, 2025

Declaration of Originality

I, **Parvesh Barak**, with SR No. **06-18-01-10-22-23-1-23181** hereby declare that the material presented in the thesis titled

An error correction algorithm for long-read sequencing

represents original work carried out by me in the **Department of Computational and Data Sciences** at **Indian Institute of Science** during the years **2023-2025**.

With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date:

Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name:

Advisor Signature

Dr. Chirag Jain

© Parvesh Barak
September, 2025
All rights reserved

DEDICATED TO

My beloved parents

who have always been my strength.

Acknowledgements

First and foremost, I would like to thank my advisor, Prof Chirag Jain, for his constant support and guidance. Apart from being an excellent researcher, he has been an excellent advisor who keeps striving for my academic excellence. I am grateful for his guidance not only in my research project but also for helping me make thoughtful decisions about my academic and professional journey ahead.

I would also like to thank my labmates, Ghanshyam, Daanish, Sudhanva and others, for the fun and open discussions. I am grateful to my friends, who made this research journey far more enjoyable than I ever imagined.

Finally, I would like to express my heartfelt gratitude to my parents, whose love and blessings have always motivated me to strive for excellence.

Abstract

Long-read sequencing technologies have transformed genomics by generating longer and sufficiently accurate DNA sequences, offering advantages in analysing highly repetitive and complex regions of a genome. A long-read sequencing machine can produce terabases of data in one day, with individual reads ranging from 10,000 to 100,000 bases in length. However, these instruments also introduce errors during sequencing. The average sequencing error rate ranges from 0.5% – 5%, depending on the choice of instrument and sequencing protocol. These errors must be corrected to enable accurate genome reconstruction. In diploid organisms such as humans, two copies of each chromosome are inherited, one from each parent. These copies, referred to as haplotypes, are nearly identical and differ at only about 0.1% of the genomic positions. These differences must be preserved for downstream genomics applications. To address this challenge, several ‘haplotype-aware’ error correction methods have been developed, which aim to correct errors within reads while preserving the haplotype-specific differences. However, existing solutions are based on either ad-hoc heuristics or deep learning approaches that require substantial computational resources and lack formal guarantees.

This thesis presents the first rigorous formulation for haplotype-aware error correction of long reads. Our approach builds on the minimum error correction framework originally developed for reference-based haplotype phasing, where the read clustering procedure is guided by the alignment of reads to a known reference genome. We extend this framework to the more challenging *de novo* setting, where no reference genome is available. We prove that the resulting formulation is NP-hard. To make our exact algorithm scale to large datasets, we introduce practical heuristics. Experiments using PacBio HiFi sequencing datasets from human and plant genomes show that our approach achieves accuracy comparable to state-of-the-art methods. A software implementation of our algorithm is available at <https://github.com/at-cg/HALE>.

Keywords

Genome assembly, phasing, clustering, overlap graph, consensus

Publication based on this thesis

Parvesh Barak, Daniel Gibney, and Chirag Jain. *Haplotype-Aware Long-Read Error Correction*. In 25th International Conference on Algorithms for Bioinformatics (WABI 2025). Leibniz International Proceedings in Informatics (LIPIcs), Volume 344, pp. 4:1-4:20, Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2025) <https://doi.org/10.4230/LIPIcs.WABI.2025.4>

Contents

Acknowledgements	i
Abstract	ii
Keywords	iii
Publication based on this thesis	iv
Contents	v
List of Figures	viii
List of Tables	xi
1 Introduction	1
1.1 Background	1
1.2 Related work	5
1.3 Contributions	6
1.4 Thesis outline	7
2 Preliminaries and problem formulation	8
2.1 Preliminaries	8
2.2 Problem formulation	9

3	Hardness result	12
4	Algorithm and implementation	20
4.1	Constructing multiple sequence alignments	20
4.2	Sampling overlaps based on sequence identity	21
4.3	Sampling informative alignment columns	21
4.4	Solving Problem 2.1	22
4.5	Correcting the target read	24
4.6	Implementation details	24
5	Results	26
5.1	Datasets	26
5.2	Evaluated algorithms	27
5.3	Evaluation metrics	28
5.4	Error correction performance	29
5.5	Comparison with naive algorithms	31
5.6	Runtime comparison	33
5.7	Evaluating the impact of algorithm parameters	34
5.7.1	Impact of threshold k	34
5.7.2	Impact of window size	34
6	Conclusion and future work	37
7	Appendix	39
7.1	Alternative algorithms to solve Problem 2.1	39
7.1.1	A brute force algorithm	39
7.1.2	A 2-approximation algorithm	39
7.2	Additional figures and tables	42
7.3	Software versions and commands	45

CONTENTS

Bibliography	47
--------------	----

List of Figures

1.1	Illustration of the majority voting procedure to correct errors. We refer to the read being considered for correction as <i>target read</i> . The above toy example contains a target read and its overlapping reads aligned with each other. The (unknown) errors in reads are highlighted in red. For each base on the target read, the bases from all overlapping reads that cover that position are used to perform majority voting, and the resulting corrected read is shown at the bottom.	3
1.2	IGV screenshot displaying raw PacBio reads (top) and corrected PacBio reads (bottom), aligned to the human reference genome (GRCh38). The reads were corrected using our HALE algorithm described in the later chapters of this thesis. Each horizontal gray bar represents an individual read aligned to the reference. Colored marks indicate deviations from the reference, either due to sequencing errors or true haplotype-specific variants. Post correction, nearly all sequencing errors are corrected. Most of the loci where the base differs from the reference even after correction represent the true biological variation preserved after correction.	4

LIST OF FIGURES

2.1	A toy example of the MSA input to our problem. MSA is computed between a substring of the target read and substrings of other overlapping reads. The target read is highlighted in red. Only those overlapping substrings are included in the MSA that have an end-to-end alignment with the substring of the target read (shown in dark grey). The above example has the target read split into four windows. In our implementation (discussed later), separate MSAs are generated for each window of the target read.	10
3.1	(Left) A directed graph. (Right) The resulting set of vectors when $M = 4$. The Hadamard codes are shown in pink. There are $M V $ vectors, each of dimension $M E $	14
4.1	Informative columns in a multiple sequence alignment. In each column of the above MSA, blue boxes represent the most frequent character from $\{A, T, C, G, -\}$ in that column. The green boxes represent the second most frequent character in a column. The pink boxes represent any other character that is neither the most nor the second most frequent. Based on the definition of informative columns, four out of six columns in the above example are informative.	22
5.1	Accuracy of reads from HG002 and <i>A. thaliana</i> sequencing datasets at $40\times$ and $60\times$ coverage. The plots compare the accuracy of raw reads with those corrected by HALE, Herro, and Hifiasm. Table 7.1 in Appendix shows the exact values plotted in Figures 5.1b - 5.1e.	30

LIST OF FIGURES

5.2	Accuracy of reads from HG002 and <i>A. thaliana</i> sequencing datasets at 40× and 60× coverage. The plots compare the accuracy of raw reads with those corrected by HALE, Naive-1, and Naive-2 algorithms. Table 7.2 in Appendix shows the exact values plotted in Figures 5.2b - 5.2e.	32
5.3	Read accuracy after applying HALE error correction on chr9 HG002 dataset with 60× coverage. Figures 5.3a–b illustrate the effect of varying the threshold k , while Figures 5.3c–d show the effect of using different window sizes w on read accuracy. . . .	36
7.1	Accuracy of sequencing reads from HG002 chromosomes 18 and 13, at 40× and 60× coverage. The plots compare the accuracy of raw reads with those corrected by HALE, Herro, and Hifiasm.	43

List of Tables

5.1	Read lengths and read counts of the four datasets used in the benchmark. <i>N50</i> is the length of the read such that at least half of the total bases are in reads of that length or longer.	27
7.1	The exact values plotted in Figures 5.1b - 5.1e.	42
7.2	The exact values plotted in Figures 5.2b - 5.2e.	44
7.3	Versions of software used in our experiments.	45
7.4	Commands used to run various tools.	46

Chapter 1

Introduction

1.1 Background

A genome sequencing instrument produces small fragments of the genome called *reads*. Each read is a string of characters from the DNA alphabet $\{A, T, C, G\}$. The length and the accuracy of reads depend on the sequencing technology. Long-read DNA sequencing, also known as third-generation sequencing, can generate reads that are much longer than those generated by earlier short-read sequencing methods. Short-read sequencing generates reads of length 150 bases, whereas long-read technologies can produce reads with lengths averaging over 10,000 bases, and in some cases, reaching lengths of up to a million bases [14, 39]. Longer reads are particularly desirable because they provide more context across the genome. This makes it easier to distinguish between regions that appear similar due to repetitive sequences. *De novo* reconstruction of the first complete human genome sequence in 2022 is an example of the advantage that long reads offer [27]. However, this increase in read length comes with a higher error rate. While short reads typically exhibit an error rate below 0.1%, the error rates for long reads range from 0.5% to 5%, depending on the sequencing platform and protocol used.

There are two major commercial providers for long read sequencing technol-

ogy: Oxford Nanopore Technology (ONT) and Pacific Biosciences (PacBio). Both technologies rely on real-time signal measurements. An ONT instrument measures changes in electrical current as DNA molecule passes through a nanopore [4]. A PacBio instrument monitors light pulses from DNA synthesis using zero-mode waveguides [11]. These signals are noisy and complex to decode accurately, hence sequencing reads often contain errors. Each error occurs in the form of insertion, deletion, or substitution of a base. Long reads are known to have insertions and deletions as their predominant error type [9, 39]. ONT reads are typically longer than PacBio reads, but contain a higher error rate of around 1% to 5% when compared to PacBio’s error rate which is less than 0.5% [19]. Long-read errors are more likely to occur in homopolymer regions (stretches of the same nucleotide, e.g., AAAAAA). Long-read sequencing technologies often have difficulty accurately counting the number of repeated bases, leading to insertion and deletion errors in such regions.

Errors in long reads can significantly affect downstream tasks such as *de novo* assembly of the genome sequence. Error correction is typically the first step in most genome assembly software pipelines because it improves the quality of the input data for all subsequent computations. Consequently, much attention has been directed toward correcting errors in long reads using a variety of algorithms.

Clearly, any error correction algorithm must leverage redundancy in sequencing data because each base of the target genome is sampled by several reads during genome sequencing. An obvious solution is to consider all-vs-all approximate string overlaps among reads, and subsequently correct each read by computing a consensus (majority vote) among the bases of reads that overlap it. Figure 1.1 shows a toy example to illustrate this naive technique. The key idea behind this approach is that the errors introduced by sequencing instruments are somewhat random, which means they can be corrected by comparing

overlapping reads and identifying consistent patterns.

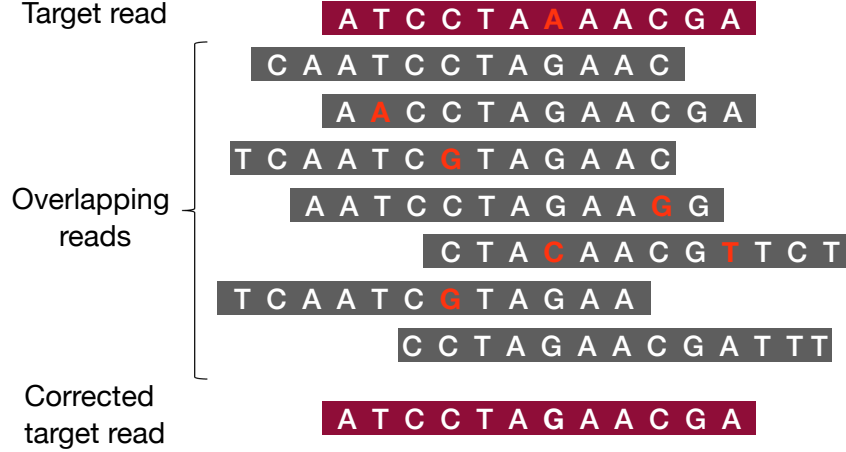


Figure 1.1: Illustration of the majority voting procedure to correct errors. We refer to the read being considered for correction as *target read*. The above toy example contains a target read and its overlapping reads aligned with each other. The (unknown) errors in reads are highlighted in red. For each base on the target read, the bases from all overlapping reads that cover that position are used to perform majority voting, and the resulting corrected read is shown at the bottom.

Unfortunately, this simple approach leads to over-correction due to repeats in the genomes. Repeats are those regions of the genome which are nearly identical and differ only in a few genomic positions. The problem is even complicated for diploid genomes like human genome, where each individual carries two copies of every chromosome: one inherited from the mother and one from the father. These two copies, known as *haplotypes*, are 99.9% identical on average [13]. The differences at the remaining 0.1% positions are referred to as *heterozygous* or haplotype-specific variations. This frequently leads to two reads originating from different copies of the chromosome overlap with each other. Note that the haplotype origin of reads, i.e., whether a read is sampled from paternal or maternal haplotype, is unknown in real data.

The naive majority voting strategy in such cases can mistakenly merge reads from the two distinct parental copies, effectively collapsing their differences

and losing important genetic information. Figure 1.2 is an IGV (Integrative Genomics Viewer) screenshot comparing raw (uncorrected) PacBio reads and corrected reads. The figure shows a 79 kb region of the human genome which contains nine heterozygous sites. A haplotype-aware error correction algorithm should retain both alleles at these sites.



Figure 1.2: IGV screenshot displaying raw PacBio reads (top) and corrected PacBio reads (bottom), aligned to the human reference genome (GRCh38). The reads were corrected using our HALE algorithm described in the later chapters of this thesis. Each horizontal gray bar represents an individual read aligned to the reference. Colored marks indicate deviations from the reference, either due to sequencing errors or true haplotype-specific variants. Post correction, nearly all sequencing errors are corrected. Most of the loci where the base differs from the reference even after correction represent the true biological variation preserved after correction.

Long reads sampled from polyploid genomes, which contain more than two near-identical copies of each chromosome, are even more challenging to correct. As a result, more sophisticated error-correction techniques are necessary to accurately distinguish between similar reads originating from different genomic regions. Furthermore, modern high-throughput instruments such as ONT’s PromethION can generate up to 13 terabases of data in a single run. As a result, any algorithm designed for error correction must not only be accurate but also scalable to handle such large volumes of data efficiently.

1.2 Related work

Long-read assemblers that support diploid genome assembly [1, 6, 7, 5, 25, 26, 29] incorporate haplotype-aware error correction as a mandatory step in their pipelines. Haplotype-aware error correction is particularly crucial for assembling diploid and polyploid genomes. Once most of the errors are corrected, one can consider approximate suffix-prefix string matches among reads using a small edit distance threshold to find a read layout for genome assembly. In contrast, skipping error correction necessitates using much higher edit distance thresholds to find correct overlaps, but this also increases the risk of capturing false overlaps between reads from different haplotypes or repetitive regions. Therefore more recently, dedicated tools for haplotype-aware error correction [21, 34] have also been developed.

Most error correction algorithms begin by computing all-vs-all overlaps between reads. For instance, given a target read to be corrected, Hifiasm [6] considers the alignments of overlapping reads to that target read. It uses heuristics to identify heterozygous loci in the target read. To preserve haplotype-specific variation, it selects only those overlapping reads that agree with the target read at all heterozygous loci. Hifiasm then applies majority voting among the selected overlapping reads and the target read for error correction. Herro [34] is a dedicated error correction tool that uses a supervised learning model, implemented using a neural network. The network is composed of convolutional blocks and a transformer encoder to distinguish sequencing errors from true variants. PECAT [25] uses a heuristic strategy that involves construction of a *partial-order alignment graph* [17] of the target read and its overlapping reads. The authors propose heuristics to filter overlapping reads sampled from the same haplotype as the target read. DeChat [21] also adopts a heuristic-driven method, performing an initial pre-correction using a de Bruijn graph, followed by a variant-aware multiple sequence alignment strategy adapted from Hifiasm

[6]. Notably, Herro [34] and Hifiasm [6, 7, 5] represent the current state-of-the-art for haplotype-aware error correction.

There also exist many long-read error correction algorithms that do not aim to preserve haplotype-specific differences [2, 23, 31, 32, 36, 38]. Most of these algorithms were developed before 2020 when the error rate in long reads ranged from 5% to 15%. This error rate was too high to distinguish errors from true haplotype-specific variants occurring at a rate of $< 0.1\%$. We skip a discussion of these old methods for brevity and refer the reader to review articles for more details [12, 40]. Our focus in this thesis is to address the haplotype-aware error correction problem, where we need to preserve haplotype-specific variation during read correction.

1.3 Contributions

In this work, we introduce a rigorous mathematical formulation of the haplotype-aware error correction problem. Our problem formulation is motivated from a different problem in bioinformatics known as *haplotype phasing*, which identifies the mutations in the target genome that are inherited together from each parent [16, 20, 28]. For a given target read, we propose a combinatorial optimization problem aimed at selecting reads that originate from the same haplotype and genomic region. An accurate selection of such reads enables a direct majority voting based correction using these reads. We prove that this optimization problem is NP-hard via a reduction from the Max-Cut problem. Despite its hardness, we show that it can be practically solved for human genome datasets using a combination of simple heuristics and brute-force search. We implemented our approach in a tool named HALE (**H**aplotype-**a**ware **L**ong-read **E**rror correction) and evaluated its performance on PacBio HiFi datasets from human and plant genomes. HALE matches the accuracy of state-of-the-art deep learning and heuristic-based methods while offering a more straightforward and theoret-

ically grounded approach.

1.4 Thesis outline

This thesis is structured as follows:

- In **Chapter 2**, we discuss a related well-studied problem known as the hypercube segmentation problem [10] which serves as an inspiration for the proposed problem formulation.
- In **Chapter 3**, we prove that the proposed problem formulation is NP-hard by using a reduction from the Max-Cut problem.
- We present our proposed algorithm, the heuristics employed to solve the problem within a reasonable time, and the implementation details of our tool HALE in **Chapter 4**.
- **Chapter 5** includes the empirical results of our algorithm. We describe the benchmarking methodology and show a detailed comparison with the baseline methods.
- **Chapter 6** offers a discussion of the findings and outlines potential directions for future research.
- **Chapter 7** serves as an Appendix and includes supplementary results. We also provide the software versions and commands to reproduce our experiments.

Chapter 2

Preliminaries and problem formulation

2.1 Preliminaries

Our proposed problem formulation for error correction of long reads is inspired by the *Hypercube 2-segmentation (H2S) problem*. The H2S problem was originally introduced by Kleinberg, Papadimitriou and Raghavan in 1998 for bi-clustering [15]. The problem statement is as follows.

H2S Problem. Given a set of n vectors $x_1, x_2, \dots, x_n$ in $\{0, 1\}^d$, one needs to select two centers c_1 and c_2 in $\{0, 1\}^d$ maximizing

$$\sum_{i=1}^n \max(\text{agree}(c_1, x_i), \text{agree}(c_2, x_i))$$

where $\text{agree}(x, y)$ counts on how many coordinates vectors x and y agree (which is also d minus the Hamming distance between x and y).

Another way to interpret the above problem is that we seek to partition n input vectors into two clusters such that the number of bit flips needed to make all the vectors agree within each cluster is minimum. H2S problem is known to be NP-Hard [10].

A generalization of the above problem, where each vector can have one or more consecutive wildcard symbols at both ends, is commonly used in the

context of reference-guided haplotype phasing application in bioinformatics [3, 8, 20, 28]. This generalized version of the problem is also referred to as the minimum error correction (MEC) problem. Solving the MEC problem is useful to determine the variants that are co-located on the same (maternal or paternal) haplotype. The MEC-based approach assumes the availability of a reference genome because each input vector is derived from an alignment of a read to the reference genome.

For *de novo* assembly, we cannot assume the availability of a reference genome. However, we assume the availability of a collection of overlaps between reads, which is a common first step in most long-read assemblers [24]. An overlap either implies (i) a sufficiently long approximate match between the suffix of a read and the prefix of another read or (ii) an approximate match of an entire read to a substring of another read. The overlaps identified using approximate string matching algorithms would comprise both true overlaps and false overlaps. The false overlaps arise from repetitive sequences within or across haplotypes. Accordingly, when considering a target read alongside its overlapping reads, there may exist one, two, or even multiple distinct “true read clusters”, each corresponding to different haplotypes or genomic loci.

2.2 Problem formulation

Our input is a multiple sequence alignment¹ (MSA) between a substring of a target read and overlapping substrings from other reads (see Figure 2.1 for an illustration). We defer the details of how these MSAs are constructed to Chapter 4. Suppose that the MSA comprises $n + 1$ vectors in $\{A, T, C, G, -\}^d$, where n is the number of overlapping substrings and d is the length of the MSA. We denote the vector in the MSA that represents the substring of the target read as t , and the set of all other vectors representing the overlapping substrings

¹We assume that the reader is familiar with the concept of multiple sequence alignment https://en.wikipedia.org/wiki/Multiple_sequence_alignment.

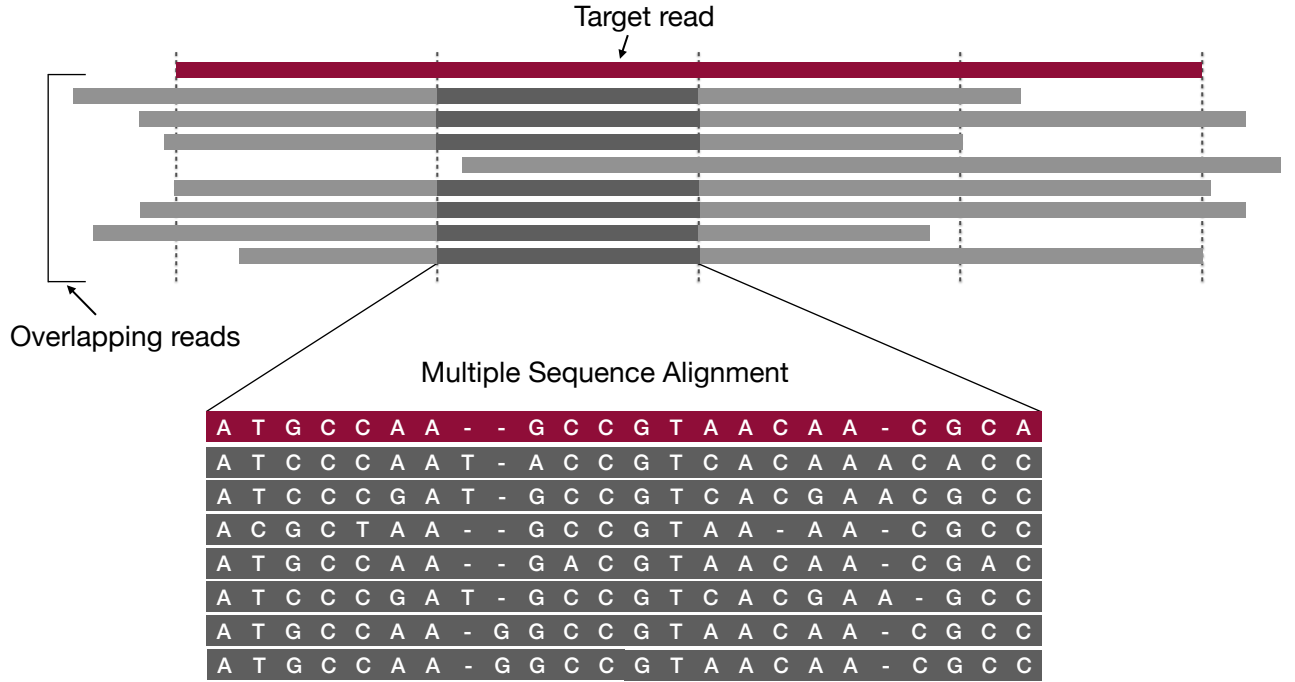


Figure 2.1: A toy example of the MSA input to our problem. MSA is computed between a substring of the target read and substrings of other overlapping reads. The target read is highlighted in red. Only those overlapping substrings are included in the MSA that have an end-to-end alignment with the substring of the target read (shown in dark grey). The above example has the target read split into four windows. In our implementation (discussed later), separate MSAs are generated for each window of the target read.

as S .

Some vectors in S represent true overlaps. They may differ from vector t due to sequencing errors in the reads. The remaining vectors differ from t due to both sequencing errors and haplotype-specific or repeat-specific variations. Accurately identifying the true overlaps is crucial for error correction, as it enables the use of a simple majority-vote strategy within the true overlaps. To achieve this, we introduce the following parsimony-based formulation aimed at selecting k vectors from S that, along with vector t , maximize a clustering score.

Problem 2.1 Given a vector t in $\{A, T, C, G, -\}^d$, a set S of n vectors in $\{A, T, C, G, -\}^d$, and a positive integer $k \leq n$, compute a subset $S' \subseteq S$ of

cardinality k and a center c in $\{A, T, C, G, -\}^d$ such that $\sum_{x \in S' \cup \{t\}} \text{agree}(c, x)$ is maximum.

We make a few remarks on this problem formulation. First, note that the symbol ‘ $-$ ’ is not given any special treatment and is processed like any other character. Second, vector c in the solution satisfies $c[i] = \text{majority}(\{x[i] \mid x \in S' \cup \{t\}\})$ for all $1 \leq i \leq d$, where ties are broken arbitrarily. Accordingly, c can be trivially computed once S' is known. After solving Problem 2.1, vector c will be interpreted as the corrected version of vector t . Third, k is a user-defined parameter in Problem 2.1. Its value should depend on the sequencing coverage. One can approximately set k to the largest value such that every genomic interval of length d is sequenced at least k times with high probability.

Our problem formulation differs fundamentally from reference-guided haplotype phasing, e.g., WhatsHap [28]. Traditional reference-guided phasing aims to reconstruct two haplotypes by partitioning reads into two clusters. This clustering is disjoint and exhaustive, i.e., every read is assigned to exactly one of the two clusters. In contrast, we focus on identifying a subset of reads that corresponds to the same haplotype and genomic region as the target read. The proposed formulation is more compatible with the *de novo* setting where the number of read clusters can vary.

Chapter 3

Hardness result

In this chapter, we prove that the proposed problem formulation for error correction in Chapter 2 is NP-hard. We establish this result via a reduction from the well-known NP-hard problem, MAX-CUT.

Theorem 3.1 *Problem 2.1 is NP-hard.*

Our proof technique is inspired by the technique used by Feige [10] to prove the hardness of the H2S problem. However, several arguments differ in our proof because we seek only one cluster in Problem 2.1 whereas the H2S problem seeks two clusters. Rather than proving the hardness of Problem 2.1 directly, we consider a related problem that slightly generalizes it.

Problem 3.1 *Given a set S of n vectors in $\{-1, 1\}^d$ and a positive integer $k \leq n$, compute a subset $S' \subseteq S$ of cardinality at least k that maximizes the ℓ_1 norm of the vector sum, i.e. $\|\sum_{x \in S'} x\|_1$.*

Later in Lemma 3.3, we will argue that Theorem 3.1 follows if Problem 3.1 is NP-hard.

Our reduction to prove the hardness of Problem 3.1 uses Hadamard codes. A Hadamard code H_M of dimension M is a collection of M vectors in $\{-1, 1\}^M$

such that every two vectors are orthogonal. There are well-known polynomial-time algorithms to construct H_M recursively when M is a power of two [35]. Therefore, we assume that M is a power of two. In the following lemma, we restate a useful property of Hadamard codes from [10].

Lemma 3.1 *Consider an arbitrary set of distinct vectors from the Hadamard code H_M . The ℓ_1 norm of their sum is at most $M^{3/2}$.*

Lemma 3.2 *Problem 3.1 is NP-hard.*

Proof: We reduce from the MAX-CUT problem. In the MAX-CUT problem, one is given an undirected graph, say $G(V, E)$, and is asked to find a subset V_1 of vertices that maximizes the number of edges between V_1 and its complement $V \setminus V_1$.

We will convert G into an instance of Problem 3.1. Let us assign an arbitrary orientation to each edge of G . We will assume an integer parameter M (we will later find an appropriate value of M which is polynomial in the size of the graph).

We build a set S comprising $n = M|V|$ vectors of dimension $d = M|E|$. For every vertex v_i of G , we introduce M vectors $v_{i,1}, \dots, v_{i,M}$ each of length $M|E|$ (see Figure 3.1 for an illustration). Each vector is divided into $|E|$ blocks of length M each. In each of the n vectors, in every block B_e associated with an edge e :

1. If v_i is head of edge e , then all entries of B_e are $+1$.
2. If v_i is tail of edge e , then all entries of B_e are -1 .
3. If v_i is not incident with edge e , then the entries of B_e in vector $v_{i,j}$ (for $1 \leq j \leq M$) are the j^{th} vector of Hadamard code H_M .

Lastly, we set the parameter $k = M(|V|/2)$.

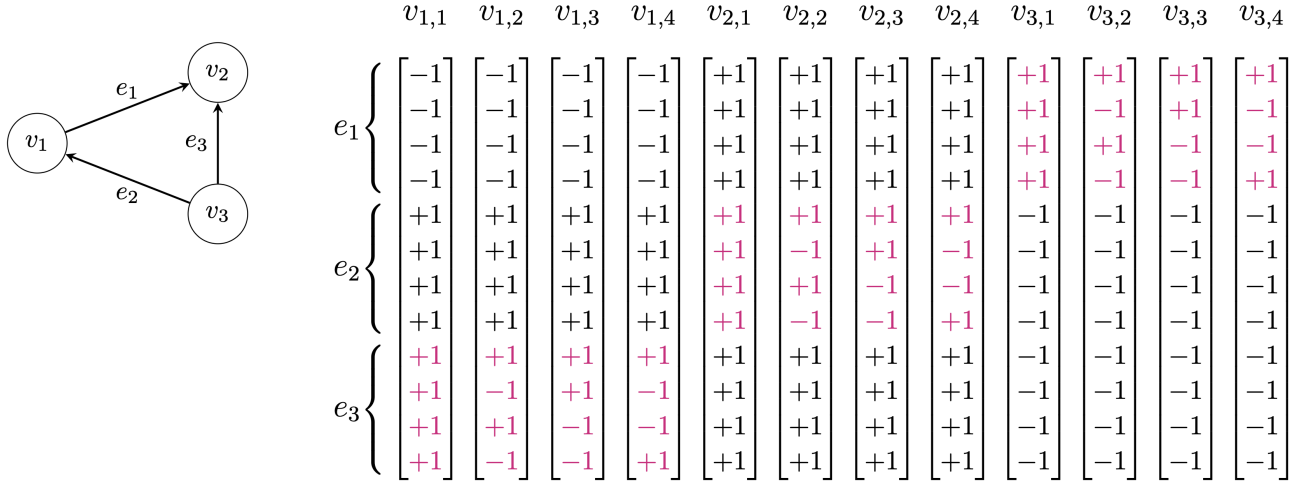


Figure 3.1: (Left) A directed graph. (Right) The resulting set of vectors when $M = 4$. The Hadamard codes are shown in pink. There are $M|V|$ vectors, each of dimension $M|E|$.

Yes instances. Suppose that $G(V, E)$ admits an optimal solution (V_1, V_2) having c edges between V_1 and V_2 . WLOG, we can assume that $|V_1| \geq |V_2|$, and hence $|V_1| \geq |V|/2$. Consider one (possibly sub-optimal) solution to the instance of Problem 3.1 that selects all $M|V_1|$ vectors from S that correspond to vertices in V_1 . Let us denote the set of these $M|V_1|$ vectors as X .

A lower bound on the ℓ_1 norm of the vector sum, i.e., $\|\sum_{x \in X} x\|_1$, can be obtained as follows. Recall that each vector in X has $|E|$ blocks. Consider a block associated with an edge e that is in the cut. There is exactly one endpoint of this edge in X , and all the M vectors corresponding to this endpoint (vertex) are monochromatic blocks of length M , which together contribute a total of M^2 to the ℓ_1 norm. This can be partially offset by other blocks. But the block of each vertex not incident with e can offset at most $M^{3/2}$ of the ℓ_1 norm (using Lemma 3.1). For an edge e , there are $|V| - 2$ vertices not incident with e . This makes the total contribution of the blocks corresponding to edge e to the ℓ_1 norm at least $M^2 - (|V| - 2)M^{3/2}$. As there are c edges in the cut, the value of this solution is at least $c(M^2 - (|V| - 2)M^{3/2})$. (The value is actually higher because the blocks associated with the edges not in the cut also contribute to

the ℓ_1 norm, but we ignore this further tightening of the bound.)

No instances. Suppose that $G(V, E)$ admits an optimal solution with $< c$ edges in the cut. Consider an arbitrary subset of vectors from S with cardinality at least k . Call this subset of vectors X_1 . This selection corresponds to a *fractional* partition (V_1, V_2) of G where the extent to which vertex v_i is in partition V_1 is equal to the fraction of its vectors in X_1 . Let z_i denote this extent for vector v_i . Similarly, let $1 - z_i$ be the extent to which the vertex v_i is in partition V_2 . For an edge $e = (v_i, v_j)$, the extent to which it is cut is $y_e = |z_i - z_j|$.

For an arbitrary edge e , the monochromatic blocks associated with its two endpoints together contribute $M^2 y_e$ to the ℓ_1 norm of X_1 . The blocks associated with vertices that are not endpoints of e , each contribute at most $M^{3/2}$ towards the ℓ_1 norm of X . Summing up over all edges and all blocks, the value of any solution is at most $\sum_e (M^2 y_e + (|V| - 2)M^{3/2}) = M^2 \sum_e y_e + (|V| - 2)|E|M^{3/2}$. Further, $\sum_e y_e \leq c - 1$. This is because it is possible to change a fractional cut (i.e., the cut corresponding to the fractional partition) into an integer cut which is at least as large (we will prove this claim below in Lemma 3.4).

Summary. If we subtract the upper bound for *no* instances from the lower bound for *yes* instances, it follows that the *yes* instance leads to an higher value than the *no* instance if

$$c \left(M^2 - (|V| - 2)M^{3/2} \right) - \left(M^2(c - 1) + (|V| - 2)|E|M^{3/2} \right) > 0$$

or $M^2 > M^{3/2} [(|V| - 2)(c + |E|)]$

Taking $M > 4|E|^2|V|^2$ suffices. Hence, finding an optimal solution to Problem 3.1 is equivalent to solving the decision version of the MAX-CUT problem.

□

Lemma 3.3 *The NP-hardness of Problem 3.1 implies NP-hardness of Problem 2.1*

Proof:

Our proof uses a chain of arguments. We begin by defining Problem 3.2, a simplified version of Problem 2.1 where we use a binary alphabet, and there is no target vector t involved in the definition. Next, we introduce Problem 3.3, a variant of Problem 3.1. To prove the lemma, we first prove Claim 3.1 to show that Problem 3.3 is NP-hard, then Claim 3.2 to establish polynomial-time equivalence between Problem 3.3 and Problem 3.2. Finally we complete the proof by proving Claim 3.3.

Problem 3.2 *Given a set S of n vectors in $\{0, 1\}^d$ and a positive integer $k \leq n$, compute a subset $S' \subseteq S$ of cardinality k and a center c in $\{0, 1\}^d$ such that $\sum_{x \in S'} \text{agree}(c, x)$ is maximum.*

Problem 3.3 *Given a set S of n vectors in $\{-1, 1\}^d$ and a positive integer $k \leq n$, compute a subset $S' \subseteq S$ of cardinality k that maximizes the ℓ_1 norm of the vector sum, i.e. $\|\sum_{x \in S'} x\|_1$.*

Claim 3.1 *The NP-hardness of Problem 3.1 (Lemma 3.2) implies the NP-hardness of Problem 3.3.*

Proof: Problem 3.3 differs slightly from Problem 3.1 because we seek a subset of size exactly k in Problem 3.3, whereas we seek a subset of size at least k in Problem 3.1. We prove Claim 3.1 using a simple polynomial-time reduction from Problem 3.1 to Problem 3.3. Given an instance of Problem 3.1 with $S = \{x_1, \dots, x_n\}$ and parameter $k = \alpha$, construct $n - \alpha + 1$ instances for Problem 3.3 using the same set of vectors S and parameter $k = \alpha + i$, for all $i \in [0, n - \alpha]$. The maximum of the solution over all instances of Problem 3.3 is the solution to Problem 3.1. $\square$

Claim 3.2 *Problems 3.2 and Problem 3.3 are polynomial-time equivalent problems.*

Proof: We consider a solution set of vectors $S' \subseteq \{0, 1\}^d$ for Problem 3.2. Observe that the vector c must satisfy $c[i] = \text{majority}(\{x[i] \mid x \in S'\})$, where ties are broken arbitrarily. Let $\phi : \{0, 1\} \rightarrow \{-1, 1\}$ be the coordinate-wise transformation defined by $\phi(0) = 1$ and $\phi(1) = -1$. For any vector $x \in \{0, 1\}^d$, let $\hat{x} = \phi(x)$ denote its image in $\{-1, 1\}^d$. Let $\hat{S}' \subseteq \{-1, 1\}^d$ be the set of vectors S' with ϕ applied to each vector. The objective function of Problem 3.2 can be written as

$$\begin{aligned}
\sum_{x \in S'} \text{agree}(x, c) &= \sum_{\hat{x} \in \hat{S}'} \frac{1}{2} \left(d + \sum_{i=1}^d \hat{x}[i] \cdot \hat{c}[i] \right) \\
&= \frac{1}{2} \left(kd + \sum_{i=1}^d \sum_{\hat{x} \in \hat{S}'} \hat{x}[i] \cdot \hat{c}[i] \right) \\
&= \frac{1}{2} \left(kd + \sum_{i=1}^d \hat{c}[i] \left(\sum_{\hat{x} \in \hat{S}'} \hat{x}[i] \right) \right) \\
&= \frac{1}{2} \left(kd + \left\| \sum_{\hat{x} \in \hat{S}'} \hat{x} \right\|_1 \right) \quad \text{since } \hat{c}[i] = \text{sign} \left(\sum_{\hat{x} \in \hat{S}'} \hat{x}[i] \right).
\end{aligned}$$

Thus, maximising the objective function of Problem 3.2 is equivalent to maximising $\left\| \sum_{\hat{x} \in \hat{S}'} \hat{x} \right\|_1$.

Given a solution to Problem 3.2, i.e., a vector $c \in \{0, 1\}^d$ and subset of vectors $S' \subseteq \{0, 1\}^d$ that maximizes $\sum_{x \in S'} \text{agree}(x, c)$, we obtain a solution for Problem 3.3 by applying $\phi(\cdot)$ component-wise to every vector in S' , creating the set $\hat{S}' \subseteq \{-1, 1\}^d$. By our earlier argument, $\left\| \sum_{\hat{x} \in \hat{S}'} \hat{x} \right\|_1$ is maximized. Conversely, given a solution for Problem 3.3 of the form $\hat{S}' \subseteq \{-1, 1\}^d$, we apply ϕ^{-1} to every vector in $\hat{S}'$ to obtain a solution $S' \subseteq \{0, 1\}^d$ for Problem 3.2. The center vector c is obtained by taking a majority over each coordinate, and the objective function of Problem 3.2 (sum of agreement with c) is maximized. $\square$

Claim 3.3 *The NP-hardness of Problem 3.2 implies the NP-hardness of Problem 2.1.*

Proof: We prove this by polynomial-time reduction from Problem 3.2 to Problem 2.1 defined on alphabet $\Sigma = \{0, 1\}$ (since this can be bijectively mapped to $\{A, T\}$). Given an instance of Problem 3.2 with set $S = \{x_1, \dots, x_n\}$ and parameter $k = \alpha$, construct n instances of Problem 2.1 where the i^{th} instance $(t^i, S^i, k = \alpha - 1)$ is constructed by considering x_i as the target vector t^i , set $S^i = S \setminus x_i$, and parameter $k = \alpha - 1$. The maximum value solution of all n instances of Problem 2.1 is the solution to Problem 3.2. $\square$

$\square$

Lemma 3.4 *Given a fractional solution to the MAX-CUT problem, there exists a rounding procedure that transforms it into an integral solution which is at least as large as that of the fractional solution.*

Proof: Let $z_i \in [0, 1]$ be the given fractional assignment of vertex v_i , for $i \in [1, |V|]$. The extent to which an edge (v_i, v_j) is cut is given by $|z_i - z_j|$. Accordingly, the cut size of the given fractional partitioning is given by $\sum_{(v_i, v_j) \in E} |z_i - z_j|$. Next, consider the following procedure to convert fractional assignment of vertices into an integral assignment. Let t be a threshold chosen uniformly at random from the interval $[0, 1]$. Assign vertex v_i to partition V_1 if $z_i > t$ and assign it to partition V_2 otherwise. For any edge $(v_i, v_j) \in E$, the probability that it is cut is given by $|z_i - z_j|$ because this occurs precisely when $z_i \leq t < z_j$. Hence, the expected value of the cut size of our integral solution is $\sum_{(v_i, v_j) \in E} |z_i - z_j|$, which is equal to the cut size of the fractional cut. This expectation guarantees the existence of at least one threshold $t^* \in [0, 1]$ for which the corresponding integral solution has value at least as large as the given fractional solution. Finally, observe that it suffices to check at only $|V| + 2$

candidate values for t : namely, at each $z_i \in \{z_1, z_2, \dots, z_{|V|}\}$, as well as the endpoints 0 and 1. This is because the cut size $\sum_{(v_i, v_j) \in E} |z_i - z_j|$ remains constant between any two consecutive values in the sorted sequence $z_1 < z_2 < \dots < z_{|V|}$. Therefore, an optimal value t^* can be found in polynomial time by checking the solution at only these $|V| + 2$ points. $\square$

Chapter 4

Algorithm and implementation

This chapter outlines the details of our error-correction algorithm, HALE. Since computing exact MSA and solving Problem 1 are both NP-Hard, we incorporate heuristics in our implementation. At the preprocessing stage, we use Minimap2 [18] software to compute overlaps among all reads. Thus, for each read, we have a set of its pairwise alignments with other reads. We perform error correction of each read independently. In the following, we refer to the read being corrected as target read.

4.1 Constructing multiple sequence alignments

To construct MSAs, we used open-source code of Herro software [34]. We briefly summarize the approach here. Herro employs the well-known star alignment heuristic, where the target read is assumed to be the ‘center of the star’. The MSA is constructed progressively, i.e., adding each overlapping read to the MSA one by one using its precomputed pairwise alignment with the target read. For further details on the star alignment heuristic, see [33].

Once the full MSA is constructed, it is divided into contiguous, non-overlapping windows of fixed length w (Figure 2.1). Each window represents an MSA of a substring of the target read and its overlapping substrings. Henceforth, each windowed MSA is processed independently. Within each window, alignments

from overlapping reads which do not fully span the window are ignored (Figure 2.1). We use the same window length $w = 4096$ as Herro.

If there are less than two alignments from overlapping reads in a window, which can happen occasionally in regions with low coverage, then Herro chops and removes the corresponding substring of the target read instead of leaving it uncorrected. We do the same in HALE. Note that this procedure may split the target read. The rationale is that the uncorrected portions of a read may negatively affect the quality of genome assembly, hence they are removed.

4.2 Sampling overlaps based on sequence identity

The count of overlapping substrings within a window is influenced by the sequencing coverage of the loci linked to the target read and how repetitive those regions are within the genome. To maintain computational efficiency of next steps, we retain only top- n overlapping substrings in each MSA. Preference is given to those overlapping substrings that have higher sequence identity¹ with the target read. If fewer than n overlaps are available, we retain all. In HALE, the default value of parameter n is set to 20.

4.3 Sampling informative alignment columns

Following the approach of Herro [34] and Hifiasm [6], we identify *informative* alignment columns in the MSA. A column is considered informative if at least two distinct characters from the set $\{A, T, C, G, -\}$ occur in that column with a minimum frequency of 3. See Figure 4.1 for an illustration. Only a small fraction of alignment columns are informative in practice. Informative columns typically indicate either biological variation among the aligned substrings or systematic sequencing errors, e.g., homopolymer indel errors. In contrast, non-informative columns are those where alignments largely agree. These columns

¹Given a pairwise alignment of two sequences, sequence identity is defined as the number of matching bases over the number of alignment columns.

have limited value for identifying true overlaps in Problem 2.1. Thus, we remove all non-informative columns from the MSA and use the modified MSA as input to Problem 2.1.

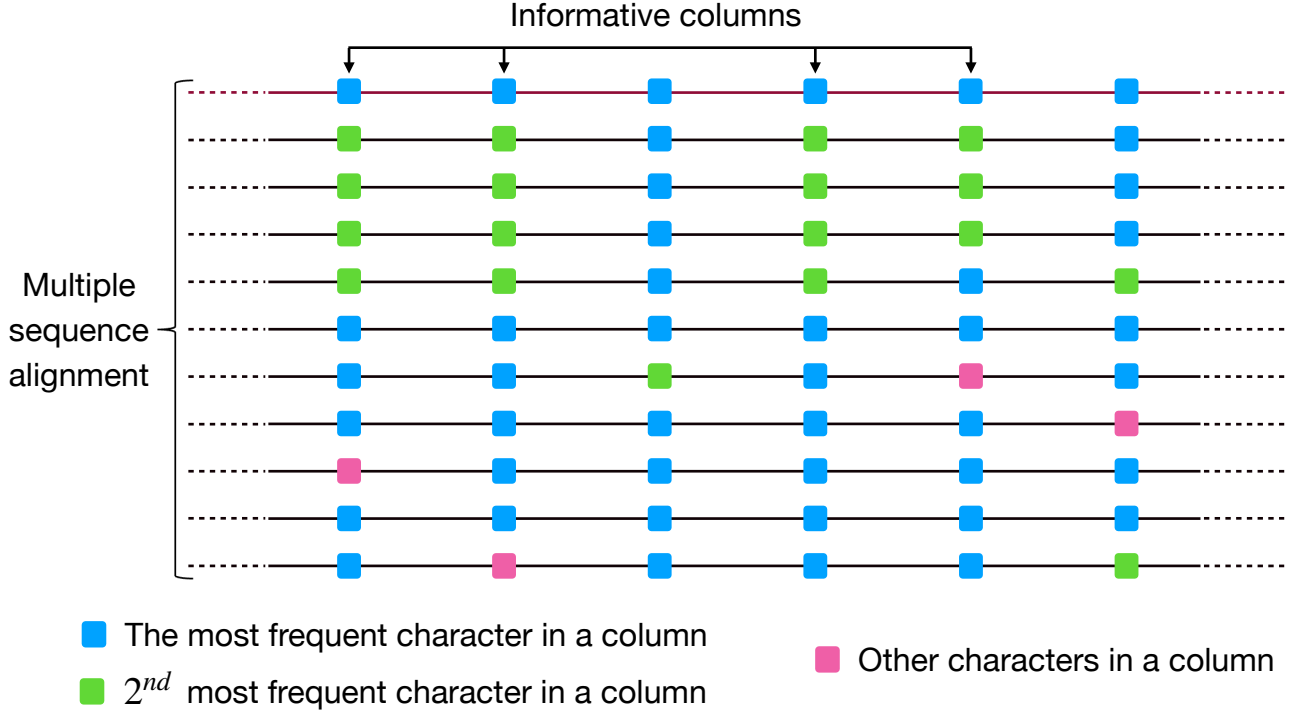


Figure 4.1: **Informative columns in a multiple sequence alignment.** In each column of the above MSA, blue boxes represent the most frequent character from $\{A, T, C, G, -\}$ in that column. The green boxes represent the second most frequent character in a column. The pink boxes represent any other character that is neither the most nor the second most frequent. Based on the definition of informative columns, four out of six columns in the above example are informative.

4.4 Solving Problem 2.1

A polynomial-time algorithm is unlikely for Problem 2.1 because we have established its hardness in Section 3. We use a direct brute-force approach to find an exact solution. In the MSA input to Problem 2.1, we have a vector t in $\{A, T, C, G, -\}^d$ (representing a subsequence of target read) along with n other vectors in $\{A, T, C, G, -\}^d$ (from the overlapping reads). Here, d denotes the number of columns in a given MSA. We solve Problem 2.1 by exhaustively

evaluating all $\binom{n}{k}$ possible ways of selecting a subset of k vectors from the total of n . For a given selection of k vectors, an optimal value of the cluster center c at the i^{th} coordinate is the character that occurs most frequently at that position among the k vectors and the target vector t . Although the runtime of this algorithm is $O\left(\binom{n}{k} \times kd\right)$, this brute-force solution is still practical due to the reduction in the size of MSA at Steps 2 and 3. In Appendix, we discuss alternative algorithms to solve Problem 2.1.

The default value of parameter k in HALE is set to $n/3$. This choice is based on our empirical observation that at least one-third of the overlapping substrings typically originate from the same haplotype as the target read (assuming ploidy ≤ 2). Choosing an appropriate value for k involves a trade-off: a significantly larger k can lead to the selection of reads with false overlaps, while a much smaller k leads to an insufficient number of reads to reliably compute a consensus.

We introduce an additional optimization. Note that Problem 2.1 treats all MSA columns with equal importance. To further improve the quality of results, we solve a weighted version of Problem 2.1 defined below. Suppose that weight function W assigns a non-negative weight $W(i)$ to each column $i \in \{1, 2, \dots, d\}$ and function $\text{match}(\alpha, \beta)$ returns 1 if characters α and β match, and 0 otherwise.

Problem 4.1 *Given a vector t in $\{A, T, C, G, -\}^d$, a set S of n vectors in $\{A, T, C, G, -\}^d$, and a positive integer $k \leq n$, compute a subset $S' \subseteq S$ of cardinality k and a center c in $\{A, T, C, G, -\}^d$ such that $\sum_{i=\{1,2,\dots,d\}} W(i) \left(\sum_{x \in S' \cup \{t\}} \text{match}(c[i], x[i]) \right)$ is maximum.*

Our motivation to introduce the weights is as follows. A majority of errors in long reads are insertions and deletions, especially in homopolymers regions [22]. Accordingly, informative columns where the most frequent and the second most frequent characters are from $\{A, T, C, G\}$ are likely to represent true biological

variation. We assign a weight 1 to such columns. In contrast, other informative columns where either the most frequent or the second most frequent character is ‘–’ may correspond to either true indel variation or sequencing artifacts. We assign a weight 1/10 to these columns. The choice of this parameter was based on our empirical observations. Developing a more principled approach for determining the weight parameters is left for future work.

4.5 Correcting the target read

Recall that before sampling alignment columns in the MSA (Step 3), we had a substring of the target read aligned with overlapping substrings from other reads, forming an MSA of length w , where w is the window length parameter. The alignment of the substring from the target read corresponds to a vector in $\{A, T, C, G, -\}^w$. We update this vector as follows. For each non-informative column, where biological variation is not expected, we replace the character with the most frequent character in that column, which is equivalent to taking a majority vote. In all informative columns, we use the optimal cluster center c , as determined by solving Problem 2.1, to update the characters. Finally, we generate the corrected version of the target read substring by removing any gap symbols (‘–’) from the updated vector.

4.6 Implementation details

We used open-source code of Herro software [34] to construct multiple sequence alignments and to identify informative alignment columns. We wrote our own implementation of the algorithm described in Section 4.4 and again used code from Herro software to do majority voting at non-informative columns. See Algorithm 1 for our pseudocode to solve Problem 2.1.

Algorithm 1 Procedure to solve Problem 2.1

Require: Target vector $t \in \{A, T, C, G, -\}^d$, set S of n vectors in $\{A, T, C, G, -\}^d$, integer $k \leq n$

Ensure: Return center vector c that minimizes the cost function in Problem 2.1

Initialize $\text{bestCost} \leftarrow \infty$

Initialize $\text{bestSubset} \leftarrow \emptyset$

for each subset $X \subseteq S$ of size k **do**

$\text{cost} \leftarrow \text{CalculateCost}(t, X)$

if $\text{cost} < \text{bestCost}$ **then**

$\text{bestCost} \leftarrow \text{cost}$

$\text{bestSubset} \leftarrow X$

end if

end for

$c \leftarrow$ empty vector of length d

for $i = 1$ to d **do**

 Compute $\text{freq}[s] \leftarrow$ number of times symbol $s \in \{A, T, C, G, -\}$ occurs at position i in all the vectors in $\text{bestSubset} \cup \{t\}$

$c[i] \leftarrow \arg \max_{s \in \{A, T, C, G, -\}} \text{freq}[s]$

end for

return c

Algorithm 2 function CalculateCost (used in Algorithm 1)

Require: Target vector t , subset S' of vectors

Ensure: Return cost

$\text{totalCost} \leftarrow 0$

$M \leftarrow S' \cup \{t\}$

$d \leftarrow$ length of each vector in M

for $i = 1$ to d **do**

for each $s \in \{A, T, C, G, -\}$ **do**

$\text{freq}[s] \leftarrow$ number of $x \in M$ with $x[i] = s$

end for

$\text{maxFreq} \leftarrow \max_{s \in \{A, T, C, G, -\}} \text{freq}[s]$

$\text{totalCost} \leftarrow \text{totalCost} + (1 + |S'| - \text{maxFreq})$

end for

return totalCost

Chapter 5

Results

In this chapter, we evaluate the performance of HALE using publicly-available PacBio HiFi sequencing datasets. The commands and software versions to reproduce the results below are available in Appendix Tables 7.3 and 7.4. The HALE source code is accessible at <https://github.com/at-cg/HALE>.

5.1 Datasets

We utilized two sequencing datasets, one from a diploid HG002 human sample¹, and the other from a haploid Col-0 *Arabidopsis thaliana* sample². As a reminder, a haploid genome contains a single haplotype, while a diploid genome contains two haplotypes (paternal and maternal). Following the benchmarking approach used in Herro [34], we restricted our evaluation to reads that are at least 10 kbp in length. Reads shorter than 10 kbp were removed.

To reduce runtime of the experiments involving the human dataset, we sampled reads from chromosome 9 of the HG002 genome. Chromosome 9 contains a high abundance of satellite DNA sequences (long arrays of tandem repeats) which makes it one of the more challenging chromosomes to assemble [27]. To isolate reads from chromosome 9, we aligned all reads to HG002v1.1 assembly³

¹<https://s3-us-west-2.amazonaws.com/human-pangenomics/T2T/scratch/HG002/sequencing/hifirevio/>

²<https://ngdc.cncb.ac.cn/gsa/browse/CRA004538/CRR302668>

³<https://s3-us-west-2.amazonaws.com/human-pangenomics/T2T/HG002/assemblies/hg002v1.1.fasta.gz>

(GCA_018852605.3, GCA_018852615.3) and retained those reads whose primary alignments mapped to either the paternal or maternal haplotype of chromosome 9. To evaluate the impact of sequencing coverage on error correction performance, we created two versions of HG002 and *A. thaliana* datasets, one with 40 $\times$ coverage and the other with 60 $\times$ coverage by randomly sampling reads. Table 5.1 gives additional information of these datasets.

Table 5.1: Read lengths and read counts of the four datasets used in the benchmark. *N50* is the length of the read such that at least half of the total bases are in reads of that length or longer.

Dataset	Number of reads	N50
chr9 HG002 HiFi 40 $\times$	562,439	18,145
chr9 HG002 HiFi 60 $\times$	915,258	18,152
<i>A. thaliana</i> HiFi 40 $\times$	354,494	15,427
<i>A. thaliana</i> HiFi 60 $\times$	531,994	15,427

5.2 Evaluated algorithms

We compared HALE against two state-of-the-art read correction tools, Herro [34] and Hifiasm [6]. Herro is a deep learning method that was primarily developed for correcting nanopore sequencing reads. Since a pretrained model for PacBio HiFi reads is not available in Herro, we used its existing model trained on Oxford Nanopore (R10) data. We assumed that the ONT-trained model generalizes to HiFi reads because HiFi reads are relatively more accurate and exhibit fewer systematic errors.

In addition to these baseline methods, we also compared HALE with two simplified correction methods. As a reminder, HALE performs a majority vote on non-informative alignment columns while using a more sophisticated technique based on solving Problem 2.1 in the informative alignment columns (Chapter 4). The naive methods employ the same heuristics described in Chapter 4, but bypass solving Problem 2.1 (Step 4).

In the first naive approach (Naive-1), a majority vote is performed across all

alignment columns, including informative ones, during Step 5. In the second naive approach (Naive-2), the original values in the aligned target read are preserved at informative columns, while a majority vote is applied only to non-informative columns. This comparison allows us to isolate the benefit of solving Problem 2.1 as an optimization step in HALE.

5.3 Evaluation metrics

For real sequencing datasets, there is no ground truth available on the errors made by the sequencing instrument. Thus, for both HG002 and *A. thaliana* datasets, we rely on high-quality references on which the reads can be mapped. For HG002 datasets, we use HG002v1.1 assembly (GCA_018852605.3, GCA_018852615.3) released by the Telomere-to-telomere consortium. For *A. thaliana* datasets, we use a high-quality assembly¹ from [37].

To evaluate read correction performance, we aligned raw reads as well as reads corrected by different methods to their respective references using Minimap2. If a read aligns to multiple regions, then only its primary alignment was considered. We used BamConcordance tool [39] to examine the alignments. BamConcordance generates an empirical log-scaled measure of base error probabilities (Q-concordance) for each read. The formula used for calculating Q-concordance is $-10 \log_{10}(1 - \text{identity})$, where identity refers to the sequence identity observed in the read alignment. As an example, Q-concordance 20 corresponds to a sequence identity of 99%, indicating that 1% of the bases are erroneous. In addition to the Q-concordance statistics, we also report the percentage of reads containing at least x mismatch (similarly, x indel) errors, for $x \in \{1, 2, 3\}$, before and after correction. In a pairwise alignment (R, Q) between a corrected read and its corresponding ground-truth sequence, the i^{th} position is a mismatch if $R[i] \neq Q[i]$ and $R[i], Q[i] \neq -$. It is an indel if $R[i] = -$ or $Q[i] = -$.

¹<https://ngdc.cncb.ac.cn/gwh/Assembly/21820/show>

As noted earlier in Chapter 4, the approach used in HALE and Herro discards a segment of a read if it cannot be reliably corrected due to insufficient overlapping read support. For a fair comparison, we evaluated only those reads that were fully corrected (i.e., not chopped) by all methods. Without this, HALE and Herro would have an unfair advantage over Hifiasm.

5.4 Error correction performance

We show a comparison of the accuracy of raw reads and those corrected by HALE, Herro, and Hifiasm, respectively, in Figure 5.1a. We achieved comparable accuracy as Herro and Hifiasm. All three correction methods led to more than a 10-fold improvement in read accuracy relative to the raw, uncorrected reads.

Next, we separately assessed mismatch and indel errors in the corrected reads. In Figure 5.1b, we highlight that the fraction of HG002 reads with one or more mismatch errors significantly drops after error correction. This result suggests that all three methods are effective in preserving biological variations in the reads. At 40× sequencing depth, HALE exhibits a marginally higher percentage of reads containing at least x mismatch errors ($x \in 1, 2, 3$) compared to Herro and Hifiasm. However, this difference diminishes at 60× depth. We also tested HALE using reads from HG002 chromosomes 13 and 18, observing a consistent trend (Appendix Figure 7.1).

In the case of *A. thaliana* dataset, which is a haploid genome sample, HALE is again comparable to the best performing method (Figures 5.1d, 5.1e). Herro lags slightly behind HALE and Hifiasm here, possibly because its model was trained using a diploid sample [34].

We also manually checked and visualized a few reads in IGV [30] where HALE failed to correct one or more errors. In some cases, we found that uneven sequencing coverage in some genomic regions resulted in MSA windows where

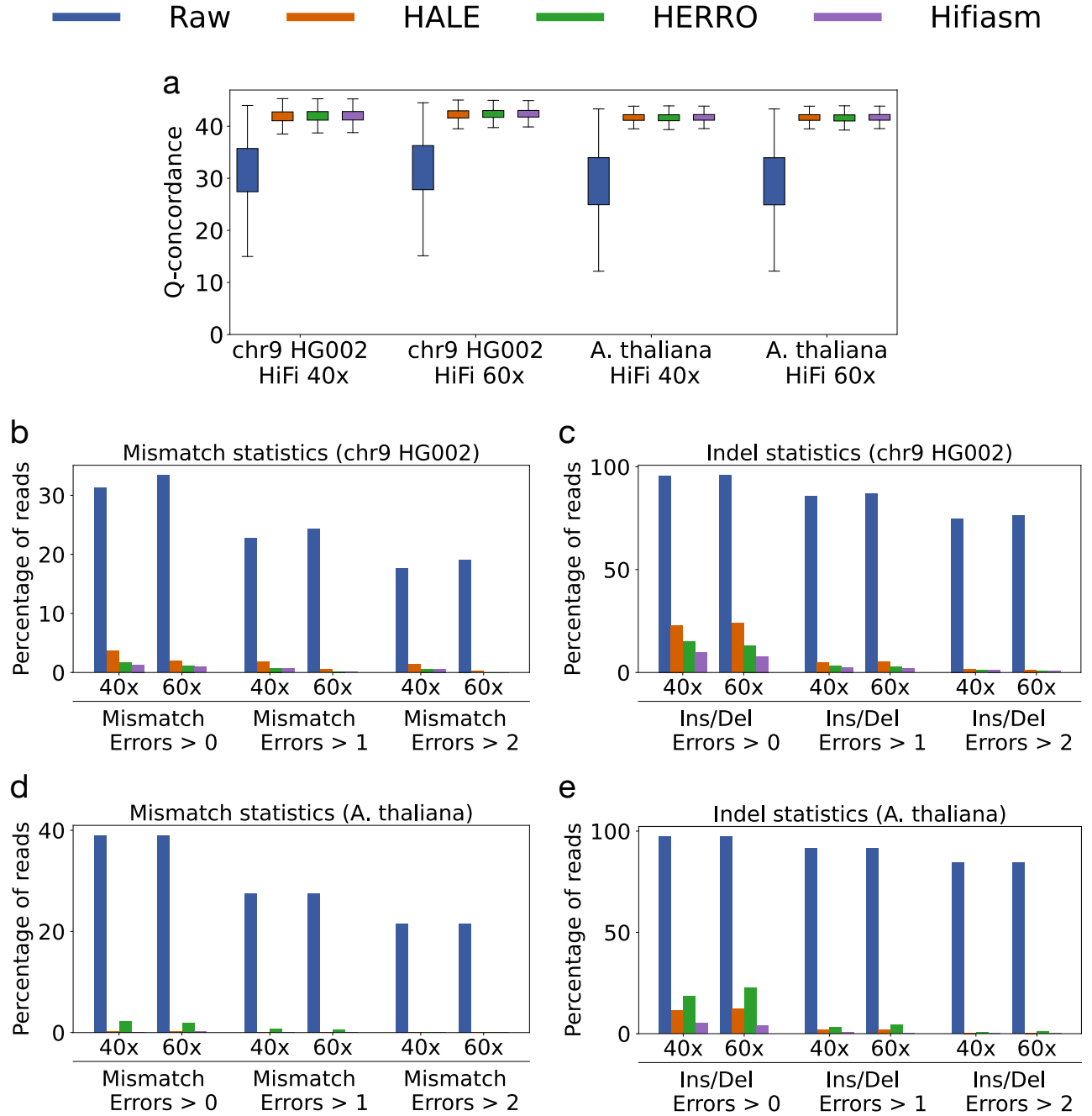


Figure 5.1: Accuracy of reads from HG002 and *A. thaliana* sequencing datasets at 40 $\times$ and 60 $\times$ coverage. The plots compare the accuracy of raw reads with those corrected by HALE, Herro, and Hifiasm. Table 7.1 in Appendix shows the exact values plotted in Figures 5.1b - 5.1e.

fewer than k true overlaps were present. As a result, our algorithm, constrained by the fixed cutoff of k , selected substrings from false overlaps for correction.

5.5 Comparison with naive algorithms

Solving Problem 2.1 (Chapter 2) is an important component of HALE as it enables us to preserve the haplotype-specific variation in reads. We compared HALE with two simplified versions of our algorithm, Naive-1 and Naive-2, respectively, which omit this optimization step.

Comparison with Naive-1: On both HG002 and *A. thaliana* datasets, the Naive-1 algorithm achieves Q-concordance scores comparable to HALE (Figure 5.2a). These results, especially on HG002 diploid datasets, surprised us initially. Recall that the Naive-1 algorithm applies the majority vote on all alignment columns, irrespective of whether they are informative or not. We further assessed the mismatch and indel error statistics. We show that HALE provides a significant improvement over the Naive-1 algorithm in correcting mismatch errors (Figure 5.2b). This confirms that a direct consensus approach eliminates haplotype variation, whereas our optimization based on solving Problem 2.1 helps to preserve them.

Despite this significant advantage in mismatch statistics, the Q-concordance scores in Figure 5.2a do not reflect a proportionate gain. This is because indel errors are the dominant error type in HiFi reads. As shown in Figure 5.2c, HALE performs comparably to Naive-1 when it comes to correcting indel errors.

Using the haploid *A. thaliana* dataset, the results are as expected. Naive-1 algorithm performs on par with HALE in terms of correcting mismatch errors (Figure 5.2d), and even slightly surpasses HALE when it comes to correcting indel errors (5.2e). A direct consensus-based approach works well on haploid datasets.

Comparison with Naive-2: We see a drop in the Q-concordance scores using the

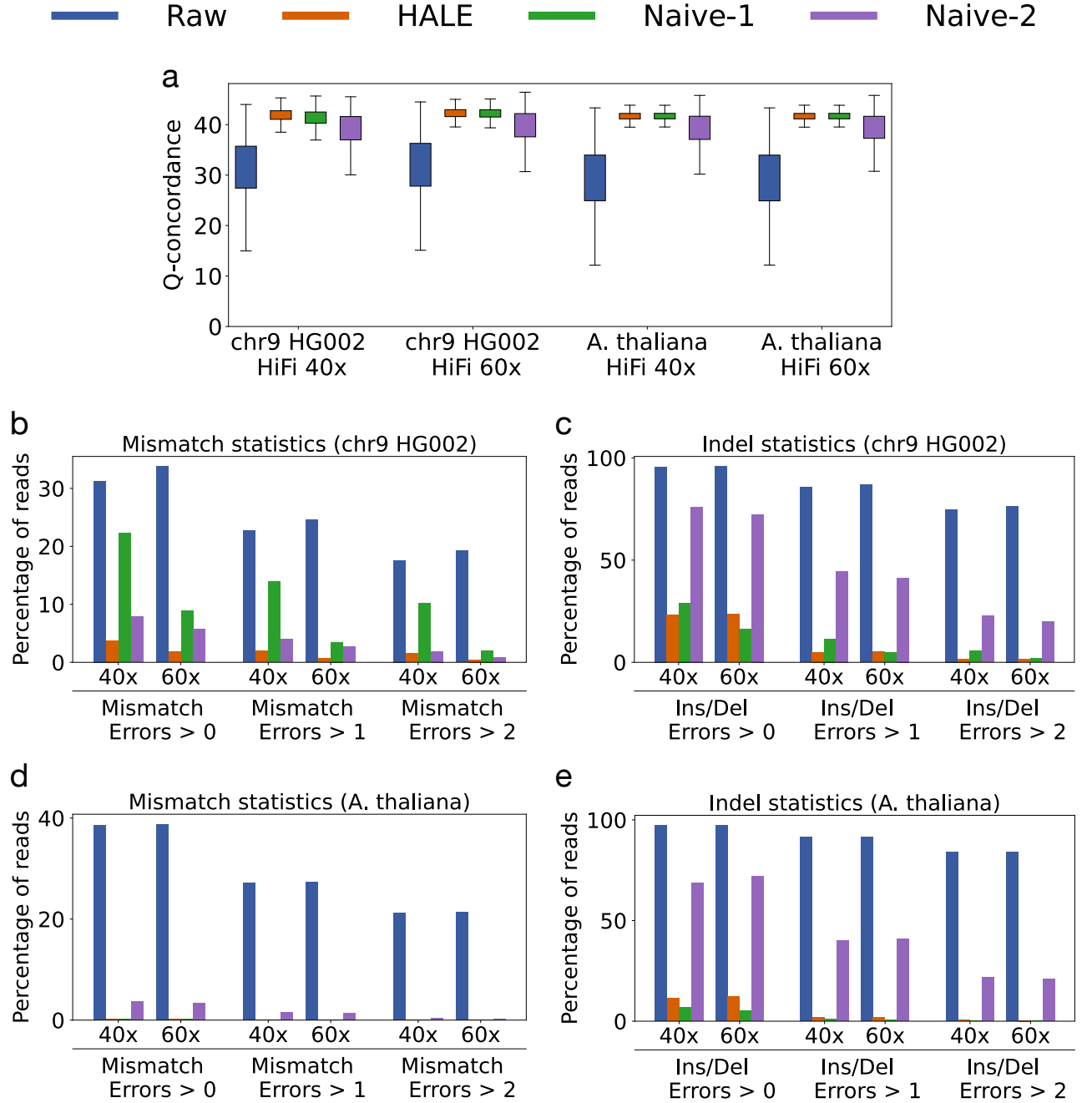


Figure 5.2: Accuracy of reads from HG002 and *A. thaliana* sequencing datasets at 40 $\times$ and 60 $\times$ coverage. The plots compare the accuracy of raw reads with those corrected by HALE, Naive-1, and Naive-2 algorithms. Table 7.2 in Appendix shows the exact values plotted in Figures 5.2b - 5.2e.

Naive-2 algorithm in both HG002 and *A. thaliana* datasets. This is because the Naive-2 algorithm makes no modification in a target read on informative alignment columns. In practice, a small fraction of informative alignment columns are classified as informative due to multiple sequencing errors occurring at the same coordinate of the target read and its overlapping reads. In such a case, Naive-2 will leave the erroneous base of the target read uncorrected. As discussed in Section 4.4, a majority of the informative sites, where either the most frequent or the second most frequent character is ‘—’ exist because of indel sequencing errors and not true biological variations. As a result, we see a much larger number of indel errors in Naive-2 (Figures 5.2c, 5.2e).

5.6 Runtime comparison

Runtime comparisons between HALE, Herro, and Hifiasm are challenging for two main reasons. First, Hifiasm uses its own custom all-vs-all read overlapping algorithm that is tightly integrated inside its code, whereas HALE and Herro use an external tool, Minimap2. Second, Herro leverages GPU acceleration, whereas both HALE and Hifiasm run on CPUs. In the following, we report the wall-clock runtimes of all-vs-all read overlapping and error correction steps for HALE and Herro. For Hifiasm, we report the total end-to-end assembly time. We report the results using chr9 HG002 60× dataset.

Minimap2 used 26 minutes to compute all-vs-all overlaps using 64 CPU threads. HALE used 9 minutes for error correction using the same number of threads. In comparison, Herro required 10 minutes on a server with four NVIDIA A100 GPUs. Hifiasm completed the full assembly in 16 minutes using 64 CPU threads, with approximately 90% of that time spent on read overlapping and error correction steps. Based on these results, we conclude that HALE is computationally more efficient than Herro. Hifiasm is the fastest overall due to its highly optimized read overlapping implementation. As all-vs-all overlap

computation is a runtime bottleneck while using HALE, using a faster read overlapper would reduce the runtime.

5.7 Evaluating the impact of algorithm parameters

The accuracy of HALE is influenced by several parameters, like number of overlapping reads n , the threshold k , and the window size w used for correction. To maintain a practical runtime across experiments, we fix $n = 20$ and based on empirical observations, we set $k = n/3$ and the window size to 4096 as default values. In this section, we explore how varying k and window size affects HALE’s performance.

5.7.1 Impact of threshold k

We evaluated four different values of k : n , $n/2$, $n/3$, and $n/8$. We show that HALE performs best in correcting mismatch errors when $k = n/3$ (Figure 5.3a) and performs worst for $k = n$ and $k = n/8$. This is expected because, setting $k = n$ is equivalent to majority voting (i.e., Naive-1). Setting $k = n/8$ leaves fewer than 3 reads in the selected subset S' , which is typically insufficient for correction when multiple reads have an error at a position. Setting $k = n/2$ may not be a good choice either, since reads from more than two genomic regions often contribute to the alignment pileup, especially in repetitive regions.

For correcting insertions and deletions errors, however, we observe a consistently decreasing trend in performance as k decreases (Figure 5.3b). This is because most informative sites involving gaps (i.e., the dash - character) are not true variant sites. In such cases, using all available reads ($k = n$) for consensus (majority voting) tends to be more reliable.

5.7.2 Impact of window size

We tested four different window sizes: 1024, 4096, 8192, and 12288. We observed that the number of mismatches increases significantly at the largest window size (12288) (Figure 5.3c). This is likely because large windows may

not be fully covered by enough overlapping reads, resulting in insufficient reads to do error correction. On the other hand, very small windows (e.g., 1024) may contain zero or too few haplotype-specific mutations sites, making it difficult to confidently find the true overlapping reads. As noted earlier, a small number of alignment columns may appear informative simply because multiple sequencing errors happen to occur at the same position across the target and overlapping reads. Having a larger number of true mutation sites helps diminish the influence of erroneous informative sites.

The trends observed for insertion and deletion errors (Figure 5.3d) are consistent with those seen for mismatches. Between the mid-range values (4096 and 8192), the performance difference is minimal, though 4096 slightly outperforms 8192 in correcting mismatch errors. As a result, we adopted 4096 as the default window size.

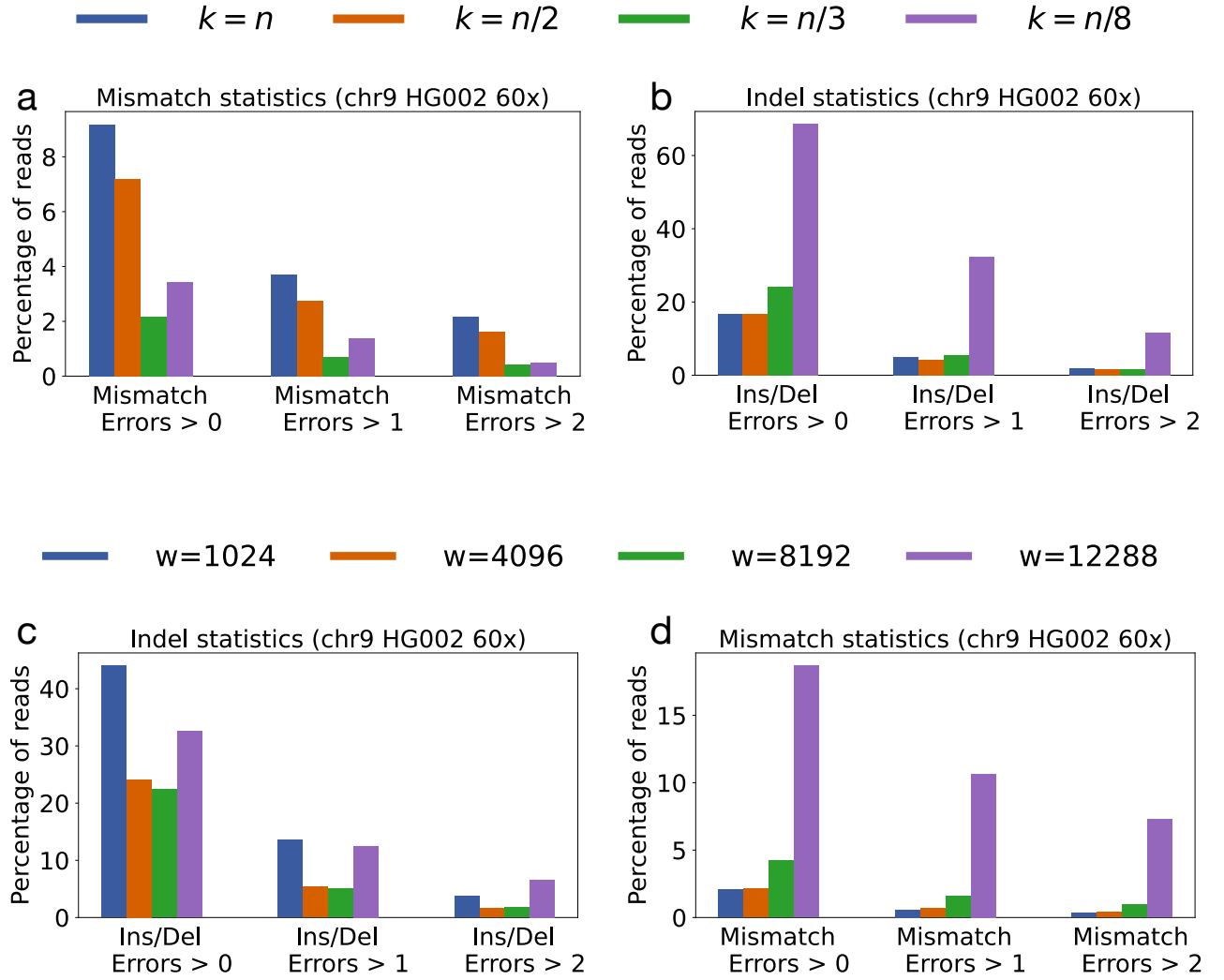


Figure 5.3: Read accuracy after applying HALE error correction on chr9 HG002 dataset with 60 $\times$ coverage. Figures 5.3a–b illustrate the effect of varying the threshold k , while Figures 5.3c–d show the effect of using different window sizes w on read accuracy.

Chapter 6

Conclusion and future work

Genome assembly tools often require a significant software engineering effort and rely heavily on heuristics that are often undocumented. It is important to investigate alternative algorithms that are grounded in solid theoretical principles. Error correction is one of the critical steps in genome assembly, especially in the context of diploid genomes, polyploid genomes, and metagenomes.

In this work, we proposed the first rigorous formulation for haplotype-aware error correction of long reads. The formulation is partly inspired by the bi-clustering formulation introduced in [15]. An extension of the same optimization framework is also used in reference-guided haplotype phasing tools such as WhatsHap [28]. Our proposed formulation is designed to identify true overlaps of a read from a given set of overlaps. Considering that the problem is NP-hard, we also discussed practical heuristics to make our algorithm scalable. We showcased the effectiveness of our algorithm using publicly available long-read datasets from a haploid *A. thaliana* genome and a diploid human genome. We showed that HALE achieves accuracy on par with state-of-the-art tools, including Herro (a deep learning-based method) and Hifiasm (which relies on intricate heuristics). Lastly, we isolated the benefit of using Problem 2.1 (Chapter 5 Section 5.5) by comparing HALE with the other simplified versions of our algorithm.

A number of improvements are possible to the proposed optimization framework as well as our implementation. One limitation of our formulation is that we independently process windows of an MSA rather than processing the entire MSA at once (Figure 2.1). This can be problematic when a window is particularly noisy or lacks sufficient haplotype-specific variation, making it difficult to select the correct subset of overlapping reads. Future work could explore alternative formulations that exploit the full length of long reads. Another limitation is our reliance on parameters such as k (count of overlapping reads to be selected) and the MSA window length. In this work, we primarily relied on our empirical observations to set these parameters. Developing either parameter-free approaches or a principled way to select these parameters which generalizes across haploid, diploid, and polyploid genomes would be an important next step.

Regarding the implementation of heuristics in HALE, the use of superior all-vs-all read aligners and MSA heuristics may further enhance read accuracy. Furthermore, it may be possible to improve the selection of informative alignment columns in an MSA such that a larger fraction of informative columns correspond to biological variation rather than sequencing errors. For example, indel variation in homopolymer regions is more likely to be a sequencing error than a biological variation [7].

HALE is theoretically promising, it is not yet a practical replacement for existing state-of-the-art tools. In future, we hope to continue improving HALE. We also hope to extend its applicability to nanopore sequencing reads, which have higher error rates than HiFi reads.

Chapter 7

Appendix

7.1 Alternative algorithms to solve Problem 2.1

7.1.1 A brute force algorithm

Problem 2.1 can also be solved by enumerating all possible center vectors and finding the best subset S'' for each center. For a given center c , the subset S'' of cardinality k that maximizes the score $(\text{agree}(t, c) + \sum_{x \in S''} \text{agree}(c, x))$ can be found trivially in time $O(nd)$, where $n = |S|$ and d is the dimension of the vectors in $S \cup \{t\}$. There are 5^d ways of choosing the center vector because the size of alphabet $\{A, T, C, G, -\}$ is five.

Accordingly, the time complexity of the above algorithm is $O(5^d \cdot nd)$. Based on our empirical observations, the value of d typically ranges between 12 and 20 in most windows. Hence this algorithm becomes impractical. This algorithm is practical in scenarios where d is small.

7.1.2 A 2-approximation algorithm

Problem 2.1 also admits a simple polynomial time 2-approximation algorithm. The approximation algorithm works by restricting the search space for the center vector to the set $S \cup \{t\}$, and identifies the best solution within this reduced search space. See Algorithm 3 for a pseudocode.

The algorithm iterates over the set of all candidate center vectors in $S \cup \{t\}$.

Algorithm 3 A 2-approximation algorithm

Require: A set of vectors S , a target vector t , and an integer k

```
1: Initialize maxScore  $\leftarrow -\infty$ 
2: Initialize bestCenter  $\leftarrow \text{None}$ 
3: Initialize bestSubset  $\leftarrow \phi$ 
4: for each vector  $s \in S \cup \{t\}$  do
5:   Find a subset  $S'' \subseteq S$ ,  $|S''| = k$ , such that the score  $\sum_{x \in S'' \cup \{t\}} \text{agree}(x, s)$  is maximized
6:   Let  $\text{score}_s$  be the corresponding maximum score
7:   if  $\text{score}_s > \text{maxScore}$  then
8:     maxScore  $\leftarrow \text{score}_s$ 
9:     bestCenter  $\leftarrow s$ 
10:    bestSubset  $\leftarrow S''$ 
11:   end if
12: end for
13: return bestCenter, bestSubset
```

For each candidate $s \in S \cup \{t\}$, it computes the subset S'' that maximizes the total score. If the resulting score exceeds the previous maximum value, the algorithm updates three variables **maxScore**, **bestCenter**, and **bestSubset** accordingly. After evaluating all candidates, the algorithm returns the center vector and the subset associated with the highest score.

For a given candidate center vector s , the subset S'' of cardinality k that maximizes the score can be found trivially in time $O(nd)$, where $n = |S|$ and d is the dimension of the vectors. Hence the time complexity of this algorithm is $O(n^2d)$.

Lemma 7.1 *Algorithm 3 is a 2-approximation algorithm for Problem 2.1.*

Proof:

We begin by reformulating the objective function of Problem 2.1 in terms of Hamming distance, as this allows us to leverage the triangle inequality property. The optimization function in Problem 2.1:

$$\text{maximize} \sum_{x \in S' \cup \{t\}} \text{agree}(x, c)$$

can be equivalently written as

$$\text{minimize } \sum_{x \in S' \cup \{t\}} d_H(x, c)$$

where $d_H(x, y)$ is the Hamming distance between vector x and y .

We will use the Hamming objective function in our proof. Suppose an exact algorithm for Problem 2.1 returns c as the center and S' as the optimal subset. Let OPT be the cost of this solution. Pick a vector s from $S' \cup \{t\}$ which has the minimum Hamming distance to the center c . Now, the cost of the solution that has s as the center and S' as the chosen subset is:

$$\begin{aligned} OPT' &= \sum_{x \in S' \cup \{t\}} d_H(x, s) \\ &\leq \sum_{x \in S' \cup \{t\}} (d_H(x, c) + d_H(c, s)) \quad (\text{by triangle inequality}) \\ &= OPT + \sum_{x \in S' \cup \{t\}} d_H(c, s) \\ &= OPT + d_H(c, s) \cdot (k + 1) \end{aligned}$$

Also $d_H(c, s) \cdot (k + 1) \leq OPT$ because s has the least Hamming distance to c among all the vectors in $S' \cup \{t\}$. Hence we have $OPT' \leq 2 \cdot OPT$.

Now, let S'' be the optimal subset of k vectors from S that minimizes the overall cost when we have s as our center vector. Let the associated cost of this solution be G . Clearly, $G \leq OPT'$. Hence $G \leq 2 \cdot OPT$.

Since, we proved that there exist a vector $s \in S' \cup \{t\}$, for which the algorithm returns a solution which is at most a factor of two away from the optimal solution. Since $S' \subseteq S$, such a vector can be found by checking all vectors in $S \cup \{t\}$ and choosing the one with minimum cost.

□

7.2 Additional figures and tables

Dataset	Error type	# Errors	Percentage of reads			
			Raw	HALE	Herro	Hifiasm
HG002 chr9 40x	Mismatch	> 0	31.3	3.7	1.6	1.3
		> 1	22.7	1.9	0.7	0.6
		> 2	17.6	1.4	0.6	0.6
	Indel	> 0	95.6	22.6	14.9	9.7
		> 1	85.8	4.9	3.1	2.5
		> 2	74.9	1.6	1.1	1.3
HG002 chr9 60x	Mismatch	> 0	33.4	1.9	1.2	1.0
		> 1	24.3	0.5	0.2	0.1
		> 2	19.1	0.3	0.0	0.0
	Indel	> 0	96.0	23.1	13.2	7.9
		> 1	86.9	5.1	2.7	2.0
		> 2	76.3	1.3	0.6	0.7
<i>A. thaliana</i> 40x	Mismatch	> 0	39.0	0.2	2.3	0.1
		> 1	27.5	0.1	0.8	0.0
		> 2	21.5	0.1	0.1	0.0
	Indel	> 0	97.4	11.5	18.3	5.4
		> 1	91.5	1.7	3.0	0.5
		> 2	84.3	0.4	0.6	0.1
<i>A. thaliana</i> 60x	Mismatch	> 0	39.0	0.2	1.9	0.2
		> 1	27.5	0.1	0.6	0.1
		> 2	21.6	0.1	0.1	0.0
	Indel	> 0	97.4	12.3	22.7	3.9
		> 1	91.5	1.8	4.5	0.4
		> 2	84.3	0.4	0.9	0.1

Table 7.1: The exact values plotted in Figures 5.1b - 5.1e.

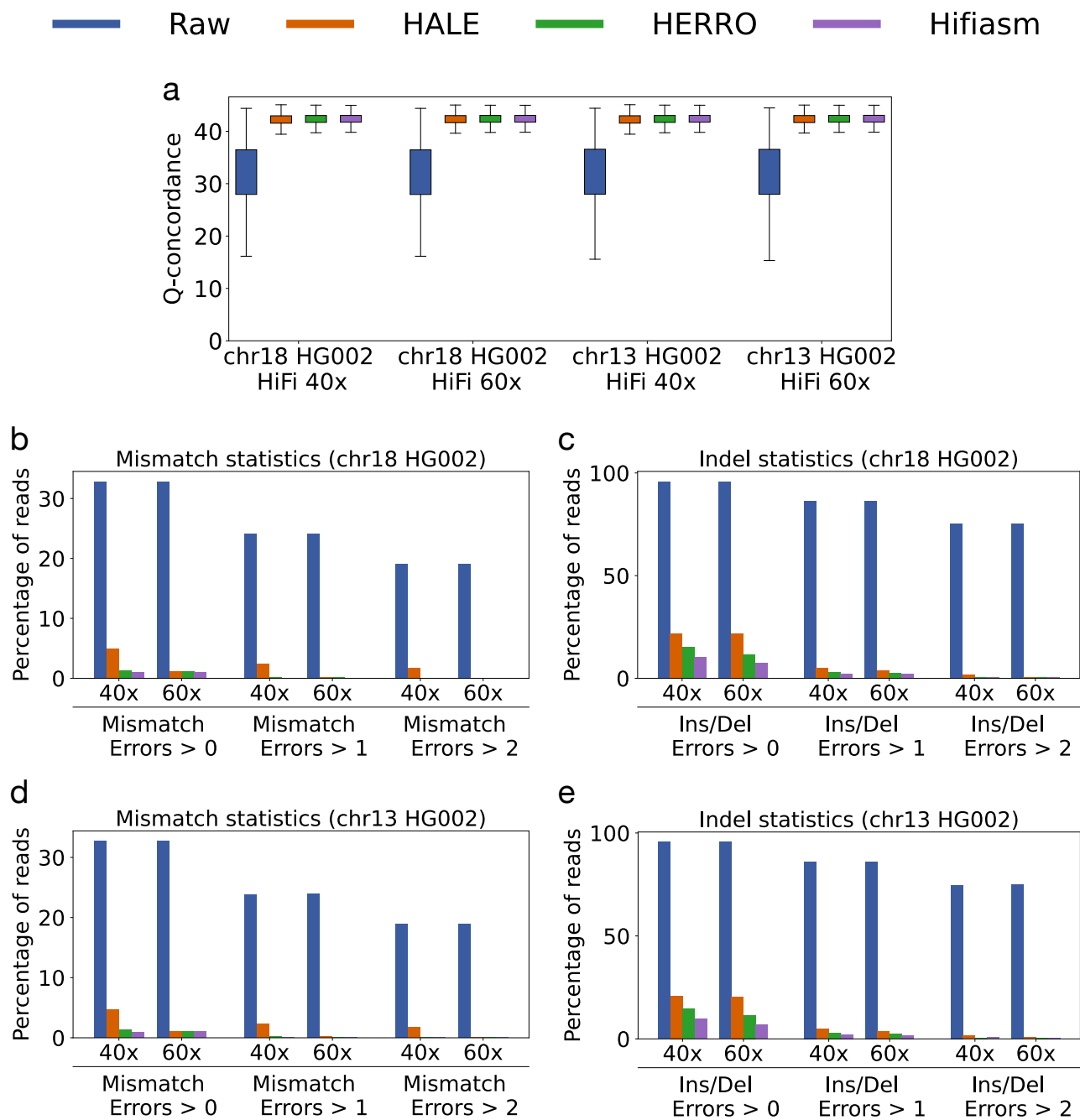


Figure 7.1: Accuracy of sequencing reads from HG002 chromosomes 18 and 13, at 40 $\times$ and 60 $\times$ coverage. The plots compare the accuracy of raw reads with those corrected by HALE, Herro, and Hifiasm.

Dataset	Error type	# Errors	Percentage of reads			
			Raw	HALE	Naive-1	Naive-2
HG002 chr9 40x	Mismatch	> 0	31.3	3.8	22.3	7.8
		> 1	22.7	2.0	13.9	4.0
		> 2	17.6	1.5	10.2	1.9
	Indel	> 0	95.6	23.1	29.0	76.0
		> 1	85.8	5.0	11.6	44.7
		> 2	74.9	1.8	5.5	22.8
HG002 chr9 60x	Mismatch	> 0	33.4	1.8	8.9	5.7
		> 1	24.3	0.7	3.5	2.7
		> 2	19.1	0.4	2.0	0.9
	Indel	> 0	96.0	23.5	16.3	72.4
		> 1	86.9	5.1	4.8	41.3
		> 2	76.3	1.6	2.0	20.0
<i>A. thaliana</i> 40x	Mismatch	> 0	39.0	0.2	0.3	3.7
		> 1	27.5	0.1	0.1	1.6
		> 2	21.5	0.1	0.1	0.4
	Indel	> 0	97.4	11.4	6.7	68.8
		> 1	91.4	1.8	1.0	40.2
		> 2	84.3	0.5	0.4	21.9
<i>A. thaliana</i> 60x	Mismatch	> 0	39.0	0.2	0.2	3.3
		> 1	27.5	0.1	0.1	1.4
		> 2	21.6	0.1	0.1	0.3
	Indel	> 0	97.4	12.2	5.3	71.8
		> 1	91.4	1.8	0.8	40.7
		> 2	84.3	0.4	0.3	21.0

Table 7.2: The exact values plotted in Figures 5.2b - 5.2e.

7.3 Software versions and commands

Tool	Version	GitHub repository
SeqKit	2.5.1	https://github.com/shenwei356/seqkit
Minimap2	2.26-r1175	https://github.com/lh3/minimap2
Samtools	1.20	https://github.com/samtools/samtools
Hifiasm	0.25.0-r726	https://github.com/chhy123/hifiasm
HALE	1.0.0	https://github.com/at-cg/HALE
HERRO	Commit: 204d29e	https://github.com/lbcb-sci/herro
Bamconcordance	Commit: dded267	https://github.com/PacificBiosciences/hg002-ccs

Table 7.3: Versions of software used in our experiments.

Software	Command
Samtools (used to extract chr9 reads)	<code>samtools view -b hg002_hifi_reads.bam chr9_MATERNAL chr9_PATERNAL samtools fastq - > chr9_hg002_hifi_reads.fastq</code>
SeqKit (read sampling)	<code>seqkit sample -p <proportion> <input_fastq> -o <output_fastq></code>
All-vs-all overlap	<code>minimap2 -K8g -cx ava-ont -k25 -w17 -e200 -r150 -m2500 -z200 -f 0.005 -t 64 --dual=yes <reads> <reads></code>
HERRO inference	<code>./herro inference --read-alns <directory_alignment_batches> -t 8 -d 0,1,2,3 -m model_R10_v0.1.pt -b 64 <fastq.gz_input> <fasta_output></code>
Hifiasm	<code>./hifiasm -t 64 -o output_prefix.asm --write-paf --write-ec temp <fastq.gz_input></code>
HALE	<code>./hale inference --read-alns <directory_alignment_batches> -m "hale" -t 64 <fastq.gz_input> <fasta_output></code>
Naive-1	<code>./hale inference --read-alns <directory_alignment_batches> -m "consensus" -t 64 <fastq.gz_input> <fasta_output></code>
Naive-2	<code>./hale inference --read-alns <directory_alignment_batches> -m "pih" -t 64 <fastq.gz_input> <fasta_output></code>
Mapping reads to reference	<code>minimap2 -ax map-hifi --eqx -t 64 --secondary=no <reference> <fasta_input> samtools view -@ 64 -S -b samtools sort -@ 64 -o <bam_output> && samtools index -@ 256 <bam_output></code>
BamConcordance (Using bam output from the mapping step)	<code>hg002-ccs/concordance/bamConcordance <fasta_input> <bam_output> <csv_output></code>

Table 7.4: Commands used to run various tools.

Bibliography

- [1] Dmitry Antipov, Mikko Rautiainen, Sergey Nurk, Brian P Walenz, Steven J Solar, Adam M Phillippy, and Sergey Koren. Verkko2 integrates proximity-ligation data with long-read de Bruijn graphs for efficient telomere-to-telomere genome assembly, phasing, and scaffolding. *Genome Research*, 35(7):1583–1594, 2025. [5](#)
- [2] Ergude Bao and Lingxiao Lan. HALC: high throughput algorithm for long read error correction. *BMC Bioinformatics*, 18(1):204, 2017. doi: 10.1186/s12859-017-1625-7. [6](#)
- [3] Paola Bonizzoni, Riccardo Dondi, Gunnar W Klau, Yuri Pirola, Nadia Pisanti, and Simone Zaccaria. On the minimum error correction problem for haplotype assembly in diploid and polyploid genomes. *Journal of Computational Biology*, 23(9):718–736, 2016. [9](#)
- [4] Daniel Branton, David W Deamer, Andre Marziali, Hagan Bayley, Steven A Benner, Thomas Butler, Massimiliano Di Ventra, Slaven Garaj, Andrew Hibbs, Xiaohua Huang, et al. The potential and challenges of nanopore sequencing. *nature biotechnology*, 26(10):1146–1153, 2008. [2](#)
- [5] Haoyu Cheng and et al. Haplotype-resolved assembly of diploid genomes without parental data. *Nature Biotechnology*, 40:1332–1335, 2022. [5](#), [6](#)
- [6] Haoyu Cheng, Gregory T Concepcion, Xiaowen Feng, Haowen Zhang, and

BIBLIOGRAPHY

- Heng Li. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature methods*, 18(2):170–175, 2021. [5](#), [6](#), [21](#), [27](#)
- [7] Haoyu Cheng, Han Qu, Sean McKenzie, Katherine R Lawrence, Rhydian Windsor, Mike Vella, Peter J Park, and Heng Li. Efficient near telomere-to-telomere assembly of nanopore simplex reads. *bioRxiv*, pages 2025–04, 2025. [5](#), [6](#), [38](#)
- [8] Rudi Cilibrasi, Leo Van Iersel, Steven Kelk, and John Tromp. The complexity of the single individual SNP haplotyping problem. *Algorithmica*, 49(1):13–36, 2007. [9](#)
- [9] Clara Delahaye and Jacques Nicolas. Sequencing DNA with nanopores: Troubles and biases. *PloS one*, 16(10):e0257521, 2021. [2](#)
- [10] Uriel Feige. NP-hardness of hypercube 2-segmentation, 2014. [7](#), [8](#), [12](#), [13](#)
- [11] Benjamin A. Flusberg, Dale R. Webster, Jessica H. Lee, Kevin J. Travers, Eric C. Olivares, Tyson A. Clark, Jonas Korlach, and Stephen W. Turner. Direct detection of DNA methylation during single-molecule, real-time sequencing. *Nature Methods*, 7(6):461–465, 2010. doi: 10.1038/nmeth.1459. [2](#)
- [12] Shuhua Fu, Anqi Wang, and Kin Fai Au. A comparative evaluation of hybrid error correction methods for error-prone long reads. *Genome biology*, 20(1):26, 2019. [6](#)
- [13] National Human Genome Research Institute. Human Genomic Variation, 2022. URL <https://www.genome.gov/about-genomics/educational-resources/fact-sheets/human-genomic-variation>. Accessed: 2025-07-13. [3](#)

BIBLIOGRAPHY

- [14] Miten Jain, Sergey Koren, Karen H. Miga, Joshua Quick, Andrew C. Rand, Tiffany A. Sasani, Jarrett R. Tyson, Andrew D. Beggs, Alexander T. Dilthey, Ian T. Fiddes, et al. Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, 36(4): 338–345, 2018. doi: 10.1038/nbt.4060. [1](#)
- [15] Jon Kleinberg, Christos Papadimitriou, and Prabhakar Raghavan. Segmentation problems. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 473–482, 1998. [8](#), [37](#)
- [16] Giuseppe Lancia, Vineet Bafna, Sorin Istrail, Ross Lippert, and Russell Schwartz. SNPs problems, complexity, and algorithms. In *European symposium on algorithms*, pages 182–193. Springer, 2001. [6](#)
- [17] Christopher Lee, Catherine Grasso, and Mark F Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002. [5](#)
- [18] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018. [20](#)
- [19] Heng Li and Richard Durbin. Genome assembly in the telomere-to-telomere era. *Nature Reviews Genetics*, 25(9):658–670, 2024. [2](#)
- [20] Ross Lippert, Russell Schwartz, Giuseppe Lancia, and Sorin Istrail. Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in Bioinformatics*, 3(1):23–31, 2002. [6](#), [9](#)
- [21] Yuansheng Liu, Yichen Li, Enlian Chen, Jialu Xu, Wenhai Zhang, Xiangxiang Zeng, and Xiao Luo. Repeat and haplotype aware error correction in nanopore sequencing reads with DeChat. *Communications Biology*, 7(1): 1678, 2024. [5](#)

BIBLIOGRAPHY

- [22] Glennis A Logsdon, Mitchell R Vollger, and Evan E Eichler. Long-read human genome sequencing and its applications. *Nature Reviews Genetics*, 21(10):597–614, 2020. [23](#)
- [23] Pierre Morisse, Camille Marchet, Antoine Limasset, Thierry Lecroq, and Arnaud Lefebvre. Scalable long read self-correction and assembly polishing with multiple sequence alignment. *Scientific reports*, 11(1):761, 2021. [6](#)
- [24] Eugene W Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85, 2005. [9](#)
- [25] Fan Nie, Peng Ni, Yadong Huang, Yujie Zhang, Yadong Wang, and Jun Wang. de novo diploid genome assembly using long noisy reads. *Nature Communications*, 15:2964, 2024. doi: 10.1038/s41467-024-40703-w. [5](#)
- [26] Sergey Nurk, Brian P Walenz, Arang Rhie, Mitchell R Vollger, Glennis A Logsdon, Robert Grothe, Karen H Miga, Evan E Eichler, Adam M Phillippy, and Sergey Koren. Hicanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. *Genome research*, 30(9):1291–1305, 2020. [5](#)
- [27] Sergey Nurk, Sergey Koren, Arang Rhie, Mikko Rautiainen, Andrey V Bzikadze, Alla Mikheenko, Mitchell R Vollger, Nicolas Altemose, Lev Ural-sky, Ariel Gershman, et al. The complete sequence of a human genome. *Science*, 376(6588):44–53, 2022. [1](#), [26](#)
- [28] Murray Patterson, Tobias Marschall, Nadia Pisanti, Leo Van Iersel, Leen Stougie, Gunnar W Klau, and Alexander Schönhuth. Whatshap: weighted haplotype assembly for future-generation sequencing reads. *Journal of Computational Biology*, 22(6):498–509, 2015. [6](#), [9](#), [11](#), [37](#)
- [29] Mikko Rautiainen, Sergey Nurk, Brian P Walenz, Glennis A Logsdon, David Porubsky, Arang Rhie, Evan E Eichler, Adam M Phillippy, and

BIBLIOGRAPHY

- Sergey Koren. Telomere-to-telomere assembly of diploid chromosomes with verkko. *Nature biotechnology*, 41(10):1474–1482, 2023. [5](#)
- [30] James T. Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S. Lander, Gad Getz, and Jill P. Mesirov. Integrative Genomics Viewer. *Nature Biotechnology*, 29(1):24–26, 2011. [29](#)
- [31] Leena Salmela and Eric Rivals. Lordec: accurate and efficient long read error correction. *Bioinformatics*, 30(24):3506–3514, 2014. [6](#)
- [32] Leena Salmela, Riku Walve, Eric Rivals, and Esko Ukkonen. Accurate self-correction of errors in long reads using de Bruijn graphs. *Bioinformatics*, 33(6):799–806, 2017. doi: 10.1093/bioinformatics/btw321. [6](#)
- [33] Joao Carlos Setubal and Joao Meidanis. *Introduction to Computational Molecular Biology*. PWS Pub. Boston, 1997. [20](#)
- [34] Dominik Stanojević, Dehui Lin, Sergey Nurk, Paola Florez de Sessions, and Mile Šikić. Telomere-to-telomere phased genome assembly using HERRO-corrected simplex nanopore reads. *Biorxiv*, pages 2024–05, 2024. [5](#), [6](#), [20](#), [21](#), [24](#), [26](#), [27](#), [29](#)
- [35] James Joseph Sylvester. LX. Thoughts on inverse orthogonal matrices, simultaneous signsuccessions, and tessellated pavements in two or more colours, with applications to Newton’s rule, ornamental tile-work, and the theory of numbers. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 34(232):461–475, 1867. [13](#)
- [36] Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 27(5):737–746, 2017. [6](#)

BIBLIOGRAPHY

- [37] Bo Wang, Xiaofei Yang, Yanyan Jia, Yu Xu, Peng Jia, Ningxin Dang, Songbo Wang, Tun Xu, Xixi Zhao, Shenghan Gao, et al. High-quality *Arabidopsis thaliana* genome assembly with nanopore and HiFi long reads. *Genomics, Proteomics & Bioinformatics*, 20(1):4–13, 2022. [28](#)
- [38] Jeremy R Wang, James Holt, Leonard McMillan, and Corbin D Jones. FMLRC: Hybrid long read error correction using an FM-index. *BMC bioinformatics*, 19(1):50, 2018. [6](#)
- [39] Aaron M Wenger, Paul Peluso, William J Rowell, Pi-Chuan Chang, Richard J Hall, Gregory T Concepcion, Jana Ebler, Arkarachai Fungtammasan, Alexey Kolesnikov, Nathan D Olson, et al. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature biotechnology*, 37(10):1155–1162, 2019. [1](#), [2](#), [28](#)
- [40] H. Zhang, C. Jain, and S. Aluru. A comprehensive evaluation of long read error correction methods. *BMC Genomics*, 21:1–5, 2020. [6](#)