METHOD FOR HAPLOTYPE PHASING WITH NATURAL LANGUAGE PROCESSING

By

Parth Abhijit Datar

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Computer Science

May 31, 2022

Nashville, Tennessee

To my grandparents, Mr. Manohar Ragunath Datar, Mrs. Smita Datar, Mr. Girdharilal Dulloo, and Mrs. Lalita Dulloo.

ACKNOWLEDGMENTS

I would like to thank Professor Maizie Zhou, PhD for advising and supporting me through this process, and Mrs. Can Luo, a fellow member of the research lab, for laying the foundation of this research. I would also like to acknowledge my parents for all that has led up to this culmination.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

This thesis is, at initial glance, a thesis steeped heavily into a specific area of computational genomics. The problems posed and the aims of this research are certainly of interest to the field of computational genomics, but they represent the application of methodologies and formulations originating closer to computer science. In this introduction, we shall explain, at a foundational level, the Structural Variant Detection problem. The information within this introduction may at times be considered common knowledge to practitioners of genomics, but not to fellow computer scientists, eliciting an explanation sufficient for an understanding of the thesis.

1.1 Genetic Background

1.1.1 DNA

DNA (deoxyribonucleic acid) is a building block of living organisms, encoding information essential for survival and propagation of said organisms. DNA molecules occur in a double helix formation, and the rungs of the helix are base pairs; these base pairs are formed of two nucleotides, complementary to each other. There are four types of nucleotides: adenine, thymine, guanine, and cytosine. Adenine and thymine are complementary, and so are guanine and cytosine. These four, though their names scarcely elicit consideration as aids to computation, shall be principal in our analysis of DNA through natural language processing.

As the double helix suggests, these base pairs exist adjacent to each other. A DNA molecule, should one have the ability to do so perfectly, may be read as a bipartite sequence of base pairs. A group of *k* base pairs is called a k-mer, and is analogous to a concept of n-grams, which we shall expound upon later. Though with perfect ability one might read in an unbroken sequence, current technology reads DNA in varying lengths. The effect of this length too shall be explored. However, with multiple separate sequences, an image of the chromosome, composed of two DNA molecules, arises in full measure.

As humans get a set of chromosomes each from their mothers and father, humans are diploid organisms. This separation means that sequences we observe in a chromosome may come from either a human's mother or father. The problem this thesis wishes to tackle is based on the possession of both a maternal and paternal haplotype amongst humans, and a desire to distinguish segments from both.

1.1.2 Structural Variants

Within either chromosome, one may detect phenomena referred to as structural variants. Structural variants are alterations to the DNA amounting to differences of greater than 50 base pairs, whose presence may be linked to a host of diseases, evolutionary differences, and phenotypical expressions (Jiang et al., 2020). Therefore, their study and, for our purpose, detection are important; tools and methodologies must be developed to this purpose. These tools must be generalizable to different sets of data, from humans to other living organisms. Furthermore, it must be scalable to the varying sizes of genomic data.

1.1.3 Short Reads

The phenomenon of next generation sequencing allowed for the sequencing of an entire human genome (Heather and Chain, 2016). Next generation sequencing, as noted by Heather and Chain, so called because is it improved upon a prior generation of sequencing, allowed for parallel computation and single nucleotide precision through measuring levels of pyrophosphate by the luminance emitted from them. Strands of DNA from 400 to 500 bp were amplified in separate clusters numbering around a million or so, providing orders of magnitude of improvement. The reads generated by such sequencing are considered short reads, as the earlier generation of sequencing used reads of around 1 kbp.

Structural variants are difficult to study with short sequences of DNA, because the structural variants are very often bigger than the sequences in which they are being searched (Heller and Vingron, 2019). When using short reads for the detection of structural variants, indirect methodologies must be used (Alkan et al., 2011). Some classes of methodologies, as noted by Alkan et al. use alignment information including discordant read pairs, split reads and read depth.

1.1.4 Long Reads

The use of long read sequences, as opposed to short reads, allows for large window for search, moving past the problem of common structural variants being larger than the sequences. To obtain such long reads, one possible method is Pacbio sequencing. In Pacbio sequencing, the intensity of fluorescent groups attached to DNA base is recorded when it is paired to a matching base, allowing parallel observation of multiple strands of DNA being formed (Rhoads and Au, 2015). Due to high error rates, multiple sequencings are performed on a molecule in order to reach consensus. Overall, data was obtained generally in excess of 10 kbp (kilobasepairs).

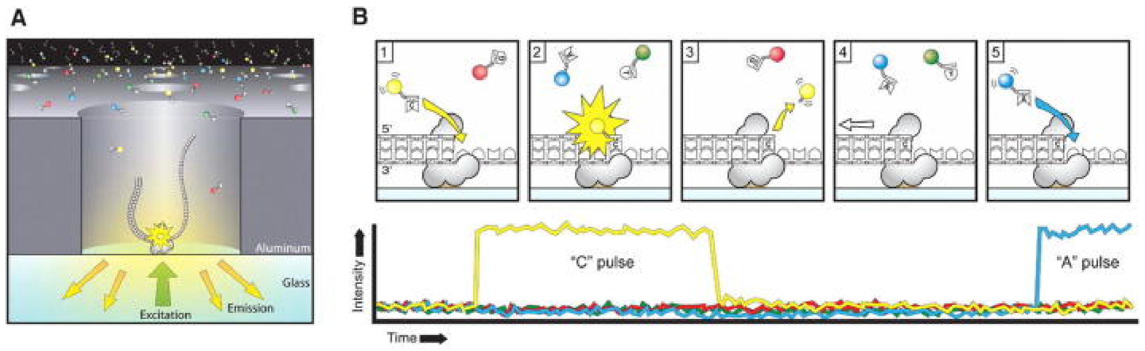

Figure 1.1: Pacbio Sequencing Demonstration
(Rhoads and Au, 2015)

1.2 Related Work

In recent years, much work has transpired in an attempt to detect structural variants. SVIM is one such tool, which uses aligned long reads (Heller and Vingron, 2019). This alignment of long reads refers to their positional mapping in accordance with a reference genome. SVIM uses these aligned long reads to detect signatures of structural variants, and clusters these signatures positionally and lengthwise. These clusters of structural variants may then be used to classify the structural variants within the corpus of reads.

CuteSV is a similar tool tailored for diploid genomes developed by Jiang et al. (Jiang et al., 2020). It boasts a higher scalability than other tools, including SVIM, and promises near linear improvement in performance relative to CPU threads. Also using structural variant signatures found in aligned long reads, it may threshold and determine the presence of structural variants on both haplotypes in an allele-wise fashion. In multiple classes of structural variants, CuteSV had an F1 score higher than its competitors which also used aligned long reads, such as PBSV (pbs, 2018) and SVIM.

As opposed to the aforementioned tools, Dipcall is a structural variant detection tool which does not use aligned long reads as the input to its pipeline (Li et al., 2018). Instead, it takes phased, assembled contigs as its input, whereby phasing refers to the separation of DNA according to its haplotype, maternal or paternal. The assembled contigs are contiguous sequences of DNA constructed from long reads. Though these contigs can be constructed from short reads, one runs into the aforementioned limitations of short reads in comparison with long reads.

SVIM-asm, another such assembly based tool, is an improvement upon SVIM, also developed by Heller and Vingron (Heller and Vingron, 2020). It operates in similar fashion to Dipcall, and SVIM-asm calls were shown by the paper to be better than DipCall, the only contemporary public alternative for diploid assemblies at the time; in structural variant calling, SVIM-asm possessed better F1 scores and detected more structural variant classes.

CHAPTER 2

Problem

2.1 Current State

As opposed to simple long reads, our current pipeline for structural variant detection uses HiFi reads. HiFi reads refer to long reads whose accuracy has been improved by circular consensus, leading to low error rates (Wenger et al., 2016).

Beginning with a corpus of HiFi reads, we may align reads to a reference genome using Minimap2 (Li, 2018) or NGMLR (Sedlazeck et al., 2018). Possessing molecular precision, we may examine differences of single nucleotides between the sequenced reads and the reference genome. Reads which originate from the same haplotype possess the same alleles given the same single nucleotide polymorphism positions. Sequencing error might hamper this, however this is infrequent in HiFi reads. Where such single-nucleotide-variants are present in one haplotype, but not the other, a separation of similarly aligned reads is possible. Longshot is a tool which uses these single nucleotide variations to phase diploid genomes (Edge and Bansal, 2019).

After phasing, local contigs may be generated for reads of the same haplotype and region. This local region is referred to as a phase block. These contigs may be generated by a tool called Hifiasm (Cheng et al., 2021). Hifiasm relies upon long, high-fidelity data to create diploid local assemblies (Wenger et al., 2016). It does so by treating reads as graph nodes, and by separating portions of the graph where two haplotypes overlap. Once this is done, common regions are replicated and attached to the distinct regions. Therefore, commonalities and distinctions between haplotypes are reflected in the contigs generated.

Upon these contigs, the aforementioned tool SVIM-asm may be used to detect structural variants (Heller and Vingron, 2020). This finally yields a file in variant call format, or VCF, which documents all detected structural variants.

2.2 Problem

Prior to the assembly of contigs, it must be noted that not all reads are phased by Longshot. Within our current pipeline, approximately 25% of reads cannot be reliably phased to either haplotype in their region. These reads are called unphased reads. Currently, these unphased reads are placed within both haplotypes for contig generation, without regard to their true haplotype.

The contigs for one haplotype, therefore, are influenced by reads belonging to the other haplotype. The elimination of this double placement would reduce noise in contig generation. The development of a phasing pipeline which may accurately assign unphased reads, in a manner which improves contig generation for structural variant calling, is the goal of this thesis. This pipeline will be built upon natural language processing techniques on an analagous representation of DNA as a natural language.

CHAPTER 3

Natural Language Processing

The methodologies of our potential solution relies heavily on concepts broadly within artificial intelligence, more specifically pertaining to natural language processing and distance-based clustering. Therefore, this chapter shall build the groundwork necessary for understanding our phasing pipeline.

3.1 Building Blocks of Genetic Language

DNA not being a manmade language, wariness may arise at the application of Natural Language Processing to it. However, as was mentioned before, DNA may be read in bipartite sequences with character representations of the base pairs which compose it. A subsequence of two adenine, one guanine, and one cytosine, in ordered fashion, may be represented as “AAGC”. Its complement, therefore, we may represent as “TTCG”. There may be some confusion as to whether this subsequence represents a word or a sentence. Though manmade languages usually have a small practical limit on the length of words, in DNA this limit is far larger.

In this thesis, we consider a long reads sequence to be a sentence, and its constituent k-mers to be words. One may note that there is no fundamental bound on the value of k insofar as the k-mer is smaller than the sequence. However, in our practical definition, for a sequence in excess of 10kbp, we use k-mers with a set value of k ranging from 4 to 128 base pairs. The notion of a fixed but scalable sliding window of k-mers applies in this context should one wish to extract all genetic words from a genetic sentence. A collections of sequences, or sentences, therefore, forms a document. Though we do not use a representation of this document in our thesis, it is possible to do so due to the component-based nature of the model.

3.2 Subword-oriented Skip-gram with Negative Sampling

The basis of the main library we use, which is FastText, is the subword-oriented skip-gram model with negative sampling (Bojanowski et al., 2017). The skip-gram model, given a word, attempts to predict surrounding words, according to a scoring criteria. The negative sampling refers to the accounting of the scores of negative samples from the dictionary, which are not surrounding words. This negative sampling allows for the model to be more precise, decreasing the likelihood of false positives. The combination of these gives way to the loss function of the model.

The scoring criteria may be formalized as:

$$s(w,c) = \sum_{g \in G_w} z_g^T v_c$$

Here G_w represents the bag of character n-grams, which will be discussed in the next paragraph, and z and v represent the word vectors for the word w and its context word c respectively (Bojanowski et al., 2017). By using the logistic loss function l, and a bag of random negative samples N_g for each word, represented by $t \in \{1, ..., T\}$, we arrive at the overall loss function:

$$\sum_{t=1}^{T} \sum_{g \in G_t} l(s(w_t, w_c)) + \sum_{n \in N_{t,c}} l(-s(w_t, n))$$

3.2.1 Character N-grams

The main modification to this skip-gram model with negative sampling by FastText, however, is the inclusion of character n-grams. A word in FastText is represented as a bag of its constituent character n-grams, meaning that the word "genome" may be represented as "<gen, eno, nom, ome, geno, enom, nome, genom, enome, genome>", for a range of 3-grams to 6-grams, which is considered standard by Bojanowski et al. for best results.

A word's representation as a real vector of dimension d, therefore, is the sum of the representations of its constituent n-grams of dimension d. In our model this would indicate that a k-mer's representation as a real vector of dimension d is dependent upon the representation of sub k-mers. Whether this functionality ought to be preserved or not shall be discussed later in this thesis.

3.2.2 Inter-sequence Influence

This skip-gram model explains our desire to separate each sequence into a sliding window of k-mers, as without that, no context words would exist for the sequence, leading to an ineffective model without meaningful connections between word embeddings. With a sliding window of k-mers, a k-mer is likely to predict certain other k-mers in its context, trained on all sentences in the dataset. Due to the reappearance of k-mers in separate sequences, the components of one sequence affect another, leading to a more interconnected model.

As long reads might have significant overlap, the constituent k-mers of two overlapping long reads would also have significant overlap. Overall, this would mean that with this read coverage, the k-mer learning is reinforced by the same context, and eventually lead to reads with significant overlap having similar word vectors. This is desirable, as long sequences of overlap are also present in certain large structural variants, while the deletion of these regions of overlap also represents a structural variant.

3.3 Clustering

Clustering is the process of grouping objects together in accordance with some relation.

3.3.1 t-SNE

The t-SNE method of visualization allows for high dimensional data, such as the data representing our sequences, to be displayed in lower dimensions while preserving the distribution of the data (van der Maaten and Hinton, 2008). Its representation relies on minimizing the Kullback-Leibler divergence, which measures the difference between two distributions of points, represented as the t-Student's distribution, hence yielding the name. The standard deviations in the distributions are derived from the neighborhood of the points, and gradient descent is used to update the embeddings. By minimizing the divergence between points in the higher dimension coordinates and the lower dimensional embedding, the nature of point neighborhoods is preserved, allowing for more efficient computation on representative data in lower dimensions.

3.3.2 Use of t-SNE Coordinates

Once we have the representation of reads in a lower dimension mathematical space through t-SNE, further clustering algorithms may be used in order to obtain information about neighborhoods of reads, or of all reads in general. Therefore, within phaseblocks, rather than simple distances calculated for a coordinate to the elements of both haplotypes, overall clusters may be calculated whereby membership within the same cluster might symbolize membership within the same haplotype. Even without this, amalgamations of simple pairwise distances might demonstrate the affinity of unphased reads to one haplotype or another within a phase block.

CHAPTER 4

Methods

4.1 Data

The dataset used for these results was a sample of HiFi reads from NA24385 (na2, 2016). For benchmarking, this dataset was aligned by NGMLR to the reference genome HG19. Results from the VCF generated by SVIM-asm were evaluated with Truvari (English et al., 2022) using the Genome In A Bottle benchmark as the ground truth (GIA, 2015).

4.2 Hardware

All computation to obtain results was performed on ACCRE, Vanderbilt's computing center. However, processing occurred on a single node, so as to allow for generalizability. To further aid generalizability, threads were maintained at the default levels, so as to not assume an overly strong processor.

4.3 Unphased Read Assignment Module

Due to the problem of unphased reads after the phasing performed by Longshot, we suggest a module for the assignment of these unphased reads to one of four candidate haplotypes. For two phase blocks, which are positional clusters of reads, we have two haplotypes each, giving rise to four candidate haplotypes for placement. Unphased reads between phase blocks must therefore be assigned to one of four of candidate haplotypes.

Each category of unphased reads and its phased counterparts give rise to a training set. These training sets may be operated upon in parallel, meaning that our proposed module assigns one set of unphased reads independently of another set of unphased reads, where the two potential phase blocks are separate. This is possible due to the fact that by read alignment, we may interpolate a read as being between two phase blocks.

The final goal of this module is generating a corpus of separate phase blocks, with all unphased reads being assigned to a phase block, and a haplotype within said phase block. These phase blocks may then be assembled into contigs and evaluated for structural variants.

The intermediate steps between intake of the training sets and the generation of a corpus of separate phase blocks are characterized by a transformation of the given training sets to a language. Every Hifi long read within a training set is represented as a sentence, and every k-mer within a read is represented as a word.

The model of the language, therefore, generates word embeddings analagous to k-mer embeddings in order to generate sentence embeddings analogous to read embeddings. These read embeddings allow for

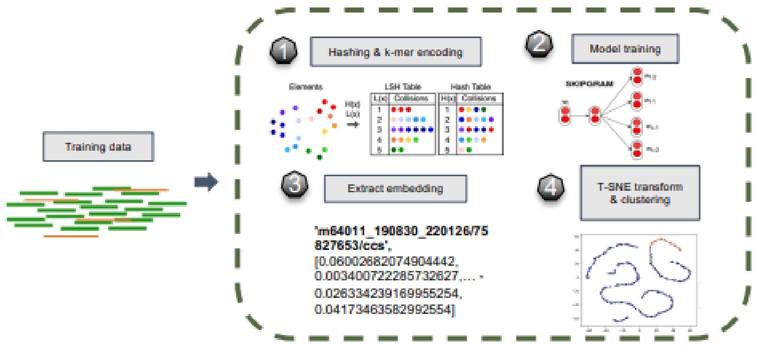

Figure 4.1: Proposed Module
Courtesy of Mrs. Can Luo

clustering that may grant every unphased read an assignment to one of the four candidate haplotypes.

Our proposed module is based heavily on the libraries of FastText (Bojanowski et al., 2017) and LSHVec (Shi and Chen, 2019). LSHVec provides the support for hashing, while FastText provides support for training the skipgram with negative sampling model for word embeddings.

As the nature of the module is modular, its subcomponents may be modified or replaced to better suit different models. Parameters of the module include k-mer length, embedding dimension, max n-grams (if appropriate), memory usage, time usage, hashing type, bucket size (if appropriate), epochs, learning update rate, and batch size.

4.3.1 Hashing

Although k-mers form the words of our language, the actual training occurs on a hashed value of every k-mer's base pair representation. The purpose of hashing within this module is to prevent space explosion due to increasing sequence length. Assume, for instance, that a toy sequence of ATGCA is given, of size 5 bytes. K-mers of length 2 would represent the sequence as <AT TG GC CA>, increasing the size of the sequence representation to 11 bytes, accounting for the spaces.

To calculate size representation, take a sequence of length n. We know that in a sequence of length n, there are $(n-k+1)$ k-mers. There are also $(n-k)$ spaces. Therefore, the total size of the k-mer representation may be calculated as:

$$(k+1)(n-k)+k$$

For a sequence of length 13000, k-mers of length 128, we obtain a size of 1660616. This would increase the size of the representation by a factor of $\frac{1660616}{13000} = 127.739692308 \approx 128$. Therefore, a corpus of Hifi reads amounting to 6 GB would transform to a corpus of k-mer representations of approximately 766 GB. As one chromosome in our genome is approximately 6 GB in length, this increase in space usage is untenable.

However, let us assume that every k-mer may be represented by an integer of fixed character length c. Then, we arrive upon a calculation of size as:

$$c(n-k+1)+n-k$$

Assuming $c=4$ with $n=13000, k=128$, we obtain a size of 64364, and a factor of $\frac{64364}{13000}=4.95107692308 \approx 5$. Here, we may represent a genome of size 6 GB as approximately 30 GB. Such a quantity could feasibly be loaded into memory, greatly increasing speed for any application that would need to load this data. Since our word embedding model is one such application, we greatly improve our training time by allowing our training data to be kept in memory as opposed to being read from disk.

Note that it is possible to store such files in binary format, reducing the space required for each base pair to 2 bits, as there are 4, therefore arriving upon an equation of $\frac{(k+1)(n-k)}{4}+k$. In the aforementioned case, this would arrive upon a representation of 415250 bytes, and a factor of $\frac{415250}{13000}=31.9423076923 \approx 32$, which is still outdone by a representation through a fixed length integer.

However, it is not possible to assign every possible k-mer a unique integer without undoing the benefits of the fixed length method. This is due to the fact that $4^k=2^{2k}$ possibilities exist for a sequence. As $2^{10}=1024 \approx 1000=10^3$, we may note that for a k-mer of length 128, we arrive upon $2^{256} \approx 10^{256/3} \approx 10^{77}$, meaning that we would need about 78 bytes for storage.

Therefore, we might reach upon a compromise of a fixed length integer as the bound for each k-mer representation, granted that k-mers represented by the same integer have a large degree of similarity. ATAT and ATAG are reasonable candidates for representation by the same integer, though ATAT and GCCA are likely not. One potential way of performing this clustering of k-mers into integer representations is through locality sensitive hashing (Gionis et al., 1999).

Locality sensitive hashing operates by creating multiple tables with random hashing functions, and discovering the nearest neighbor within some distance ε for a query point. This allows for local comparison of points, reducing the complexity of determining what points are similar. Points given the same bucket identification are considered similar.

With k-mers as points, through locality sensitive hashing, we may give each k-mer an identification such that all k-mers which possess this identification are similar. This condensing of the k-mer space maintains relations while reducing space required greatly. This reduction of space is guaranteed by the parameter of bucket size in locality sensitive hashing, which controls the number of buckets to which points may be mapped.

For our module, every k-mer within every read, in a sliding window fashion, outputs its bucket identification in the hash file. The same k-mer on different lines outputs the same bucket identification. These bucket identification integers form the words for the training model. As the name suggests, LSHVec is tool capable of performing locality sensitive hashing on k-mers from read sequences, producing our desired hash file (Shi and Chen, 2019). This hashfile must be produced for every training set, with its four candidate haplotypes and its unphased reads all being represented.

It is worth noting that the bucket identification does not preserve character information, merely being an integer. Furthermore, this bucket identification may change from one corpus of reads to another for the same k-mer. This would prove a hindrance should one wish to hash sections of the same data in parallel, although LSHVec does provide such functionality with a PKL file for uniformity.

Therefore, should one wish to use a hash whose characters encode information about the represented k-mer at the cost of space, and provide straightforward globalization, we propose hexadecimal hashing. Hexadecimal hashing uses the onehot encodings provided by LSHVec, but in hexadecimal format. The representations of LSHVec are {'A':'00', 'C':'01', 'G':'10', 'T':'11'}. With a mere conversion to base-10 integers, a situation as such would arise: 10100100 (GGCA) would map to 164 and 11100100 (TGCA) would map to 228. With hexadecimal hashing, however, the conversions would be A4 and E4. The hexadecimal format, as can be seen, preserves constituent 2-mers.

4.3.2 Training

Although it would be possible to represent each k-mer as a onehot vector of size corresponding to the number of k-mers, the space each vector would require would be inefficient. Furthermore, if k-mer A and k-mer B frequently appear next to each other, a simple onehot encoding fails to represent this relationship.

Given a k-mer, our expectation of the surrounding k-mers indicates our predictive ability for determining what base pairs might replace the beginning or ending of the k-mer. This would naturally be useful, as it provides us with more information regarding the base pairs of the sequence at a level above random guessing. Vectors which could represent such relations would serve as good word embeddings.

Having obtained a hash file which represents every read within a training set, relations between the different k-mers must be extracted to produce word embeddings. However, as we are operating on bucket identifications, we are more precisely representing relations between sets of k-mers, internally similar. Therefore, each word embedding encodes information about surrounding sets of k-mers.

This predictive capability of a word embedding is provided to us through the aforementioned skipgram model with negative sampling, through the library FastText. By iterating over the corpus of k-mer bucket identifications, FastText refines the word embeddings for each bucket identification. As mentioned previously it does so by maximizing cosine similarity of the embeddings of words which appear within the same window and minimizing the cosine similarity of the embeddings of words which do not appear in the same window.

Therefore, training consists of the refinement of word embeddings. Therefore, a set of similar k-mers (i.e. k-mers with the same bucket identification) has an embedding a small cosine distance away from other sets which frequently appear near it.

Due to bucket identifications not encoding character-wise information on the basis of k-mers, it is illogical to use character n-grams for training. Therefore, said character n-grams ought to be disabled when using locality sensitive hashing. However, should one use hexadecimal hashing, character n-grams may be used.

With these word embeddings, the sentence embedding follows trivially in FastText. To obtain a representation of a read in the original training set, one simply takes the mean vector of all of the bucket identification embeddings which constitute the read in the hash file. Therefore, reads which have a lot of overlap will have sentence embeddings close to each other due to the large amount of k-mer overlap shifting the mean vector.

With these sentence embeddings, which may be stored with the same indexing on the original training set, the original training set is greatly reduced in size, equal to the number of reads in the original set multiplied by the dimension of the training model. With this embedding file, clustering may be performed to assign unphased reads to one of the four haplotypes.

4.3.3 Clustering

To determine the relationship between the generated read embeddings, it is important to consider pairwise distances, so as to discover the nearest neighbors of every unphased read. This allows us to determine the closest fit out of the four candidate haplotypes for the unphased reads. To perform this pairwise evalutation, t-SNE was used (van der Maaten and Hinton, 2008).

The final result of this t-SNE is the mapping of each higher dimension read embedding to a point on a lower dimensional manifold, 2-D to be more precise. Upon these final points, a simple Euclidean distance calculation can be made for each unphased read to each phased read. This allows us to determine the nearest neighbor for every unphased read from all four candidate haplotypes.

This process is more efficient following t-SNE due to a reduction in dimension ranging from factors of 25 to 100. The module for t-SNE itself is more efficient than the standard implementation due to supporting MultiCore computation (Ulyanov, 2016).

With the nearest neighbors for each unphased read, assignment of an unphased read to the candidate haplotype of its nearest neighbor is a trivial matter. Currently, advanced methodology is not used for clustering beyond this, as t-SNE itself focuses on pairwise distances. With these assignments, all reads can be partitioned into the four haplotypes, and the module may output all necessary phase blocks for contig generation.

CHAPTER 5

Results

5.1 Variant Calling Results

The results of variant calling on multiple pipelines, courtesy of Mrs. Can Luo, are shown in Table 5.1 for the subject genome. An embedding size of 200 and a k-mer size of 15 were used as parameters. As may be noted, the proposed pipeline with our integrated module possessed the highest number of true positive calls for structural variants, alongside the highest F1 score.

This modified pipeline also displayed better results than the original pipeline, validating our belief that through proper assignment of unphased reads, structural variant calling would improve. Our remaining results examine more granular matters, such as optimal parameters, timing and space complexity, and hashing types.

Model	TP	FP	FN	Recall	Precision	F1
Original Pipeline	4791	760	490	0.907	0.863	0.885
Modified Pipeline	5020	477	261	0.951	0.913	0.932
CuteSV	4684	295	597	0.887	0.941	0.913
PBSV	4122	423	1159	0.781	0.907	0.839
Dipcall	4849	673	432	0.918	0.878	0.898

Table 5.1: Variant Calling Results on Whole Genome Data

5.2 Neighbor Distance Comparison

The results for m64015_190920_185703/142934028/ccs, a randomly selected sequence, are shown in Table 5.2, where we may note that phase block is relatively easy to deduce as separable. For almost every parametrization of k-mer size and dimension, the sequence's nearest neighbor belongs to phase block 1. The only parametrization for which some shred of doubt arises is the parametrization of $k = 128, d = 50$. Despite this, even that parametrization seems to hold low values for phase block 1 haplotype 1.

Within phase block 1, the results are less straightforward. For $k = 16$, the results are relatively conclusive. However, for $k = 4$, 2 parametrizations support haplotype 2 and 1 parametrization supports haplotype 1. Since the parametrization supporting haplotype 1 has a higher dimension, it may be taken to be more reliable. With this rationale of believing the highest dimension within a k-mer size, for $k = 128$, we may concur that the sequence belongs to haplotype 1.

Though the phasing results are not clear, they do possess some consensus. In 6 of the parametrizations, the sequence belongs to phase block 1 and haplotype 2. With more advanced methods of clustering, this assignment may be refined further, potentially by considering more than the nearest neighbor within each candidate haplotype.

K-mer Size	Dimension	PB1 HP1	PB1 HP2	PB2 HP1	PB2 HP2
4	50	1.746	1.564	15.580	16.714
4	100	1.319	0.983	15.030	15.132
4	200	2.032	2.232	21.270	21.984
16	50	1.347	1.151	10.984	10.704
16	100	1.130	0.731	7.147	21.538
16	200	1.594	1.179	13.320	13.146
128	50	2.673	6.297	6.162	2.238
128	100	2.249	1.701	2.731	8.107
128	200	2.275	3.565	6.639	11.689

Table 5.2: Parametrized Comparison of Nearest Neighbors for m64015_190920_185703/142934028/ccs
PB refers to Phase Block and HP to Haplotype

5.3 Parametrization Visualization

Figures 5.1 through 5.9 display the same dataset as visualized by t-SNE coordinates over different parametrizations.

For the same k-mer length, dimension size does not a priori appear to make a difference. The overall structure of the coordinates remains the same, despite shifts and bends. This helps us maintain that although global coordinates are not fixed, local clusters preserve internal structure. This also appears to refute our earlier assumption that due to greater number of dimensions, more information may be stored, leader to more accurate predictions.

For greater k-mer lengths, especially $k = 128$, a clearer separation on the basis of haplotype arises, which allows for more confidence in phasing. As the phased reads are more clearly separated, the nearest neighbor belonging to one candidate haplotype rather than another appears to be a stronger indicator for placement.

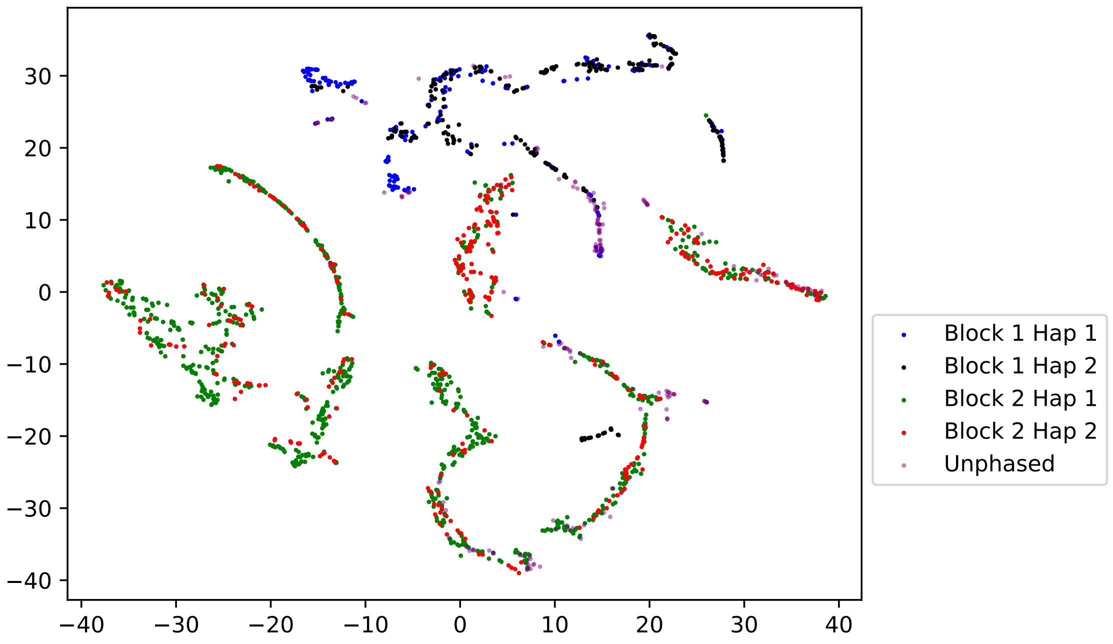

Figure 5.1: LSH with k=4, d=50

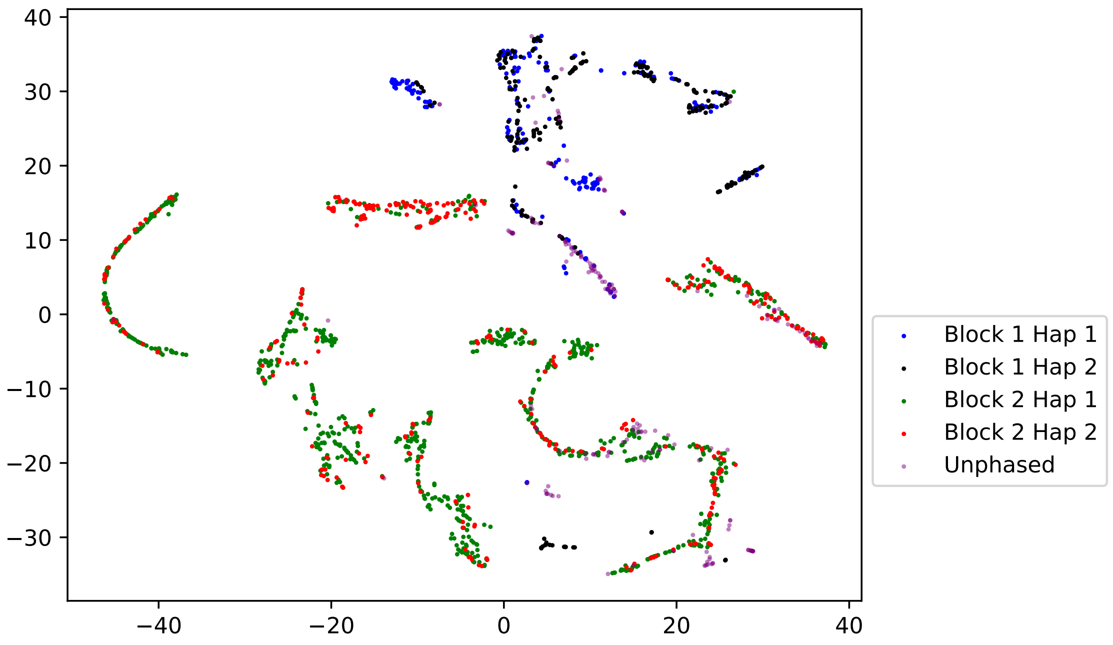

Figure 5.2: LSH with k=4, d=100

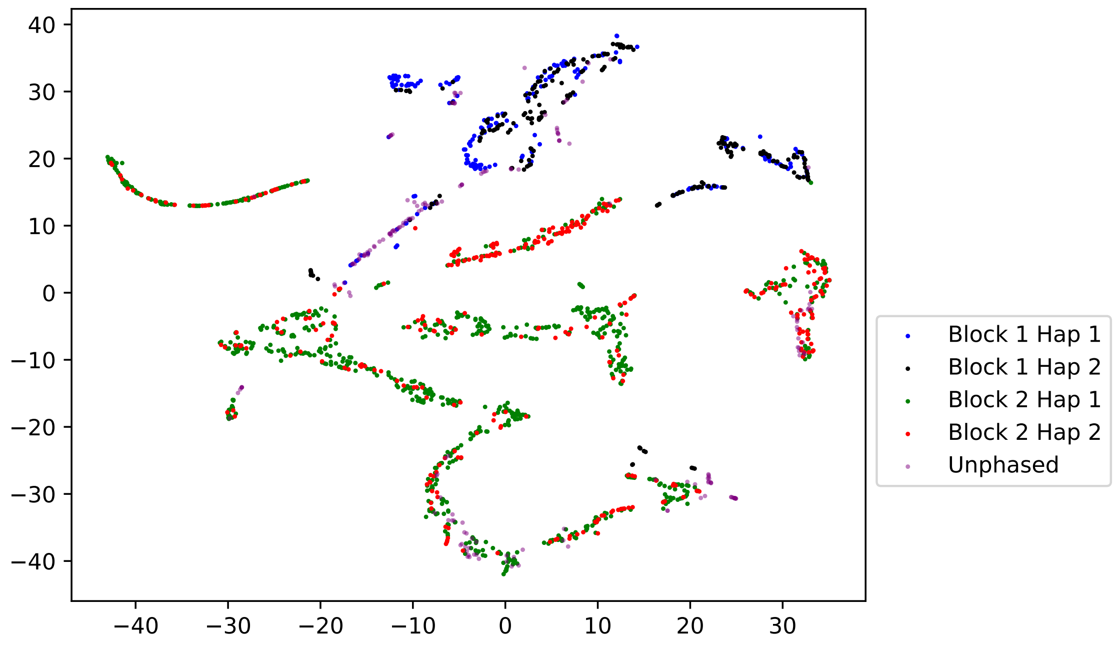

Figure 5.3: LSH with k=4, d=200

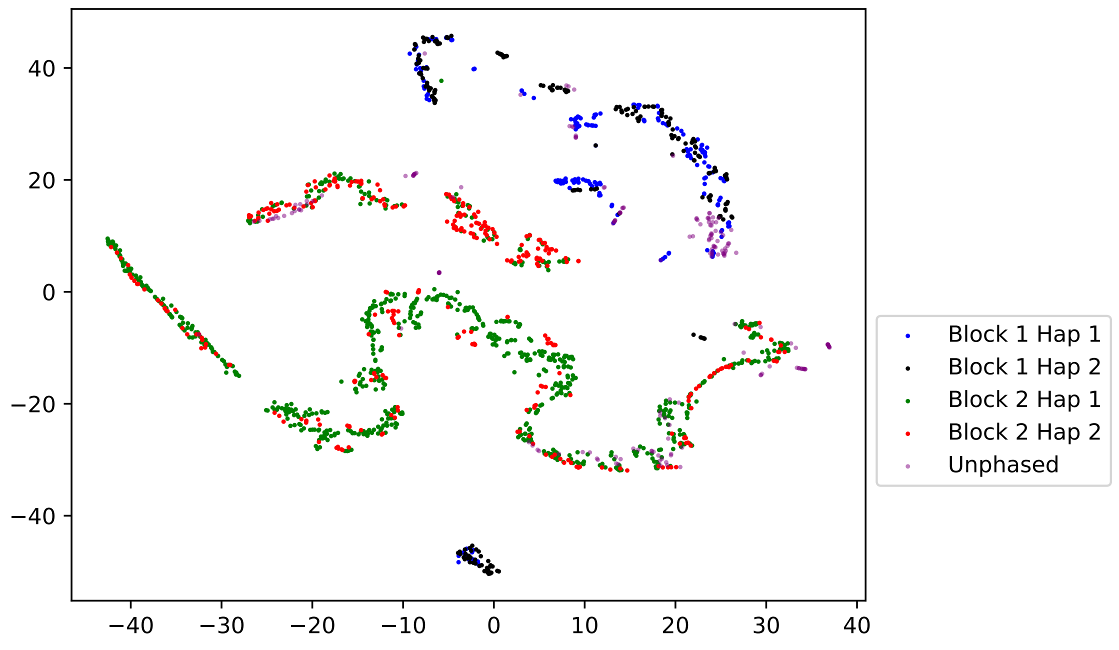

Figure 5.4: LSH with k=16, d=50

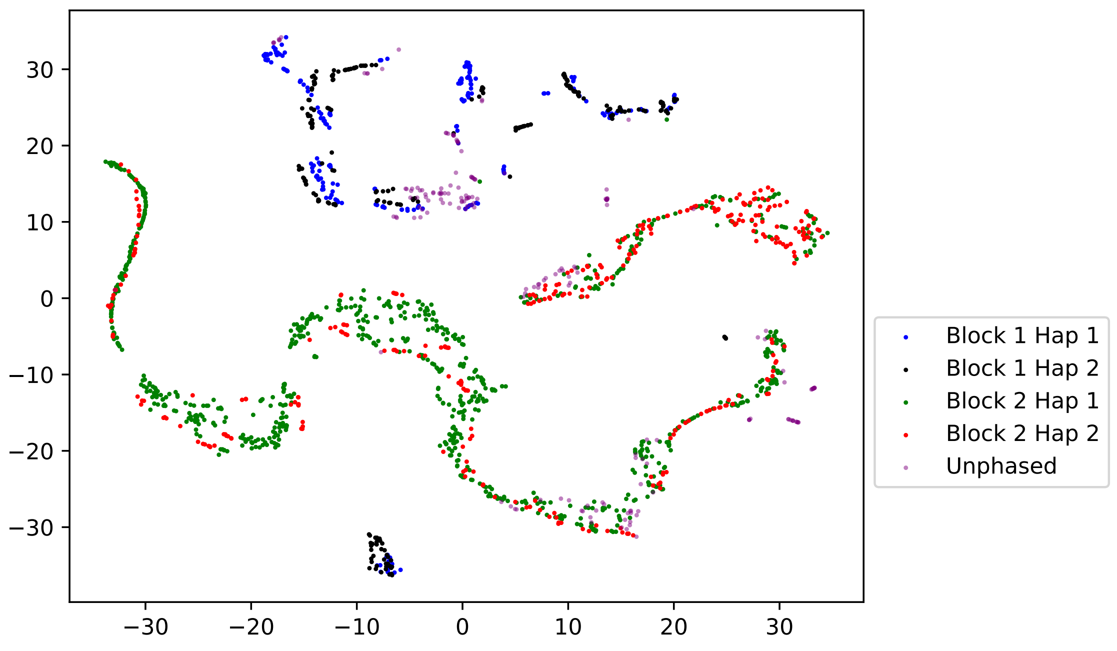

Figure 5.5: LSH with k=16, d=100

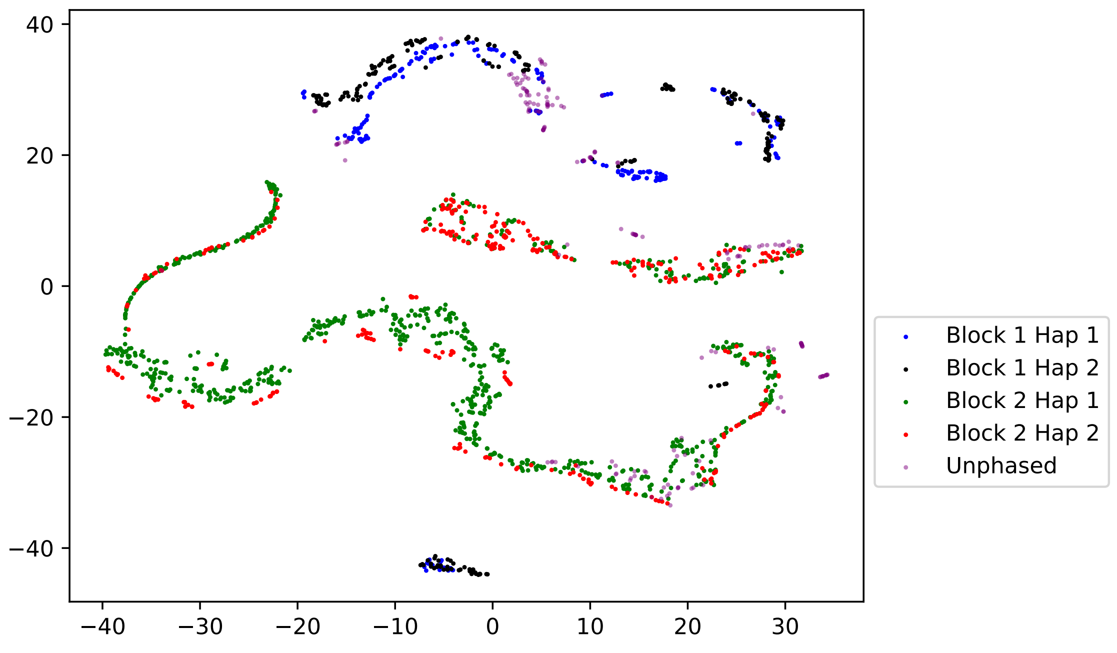

Figure 5.6: LSH with k=16, d=200

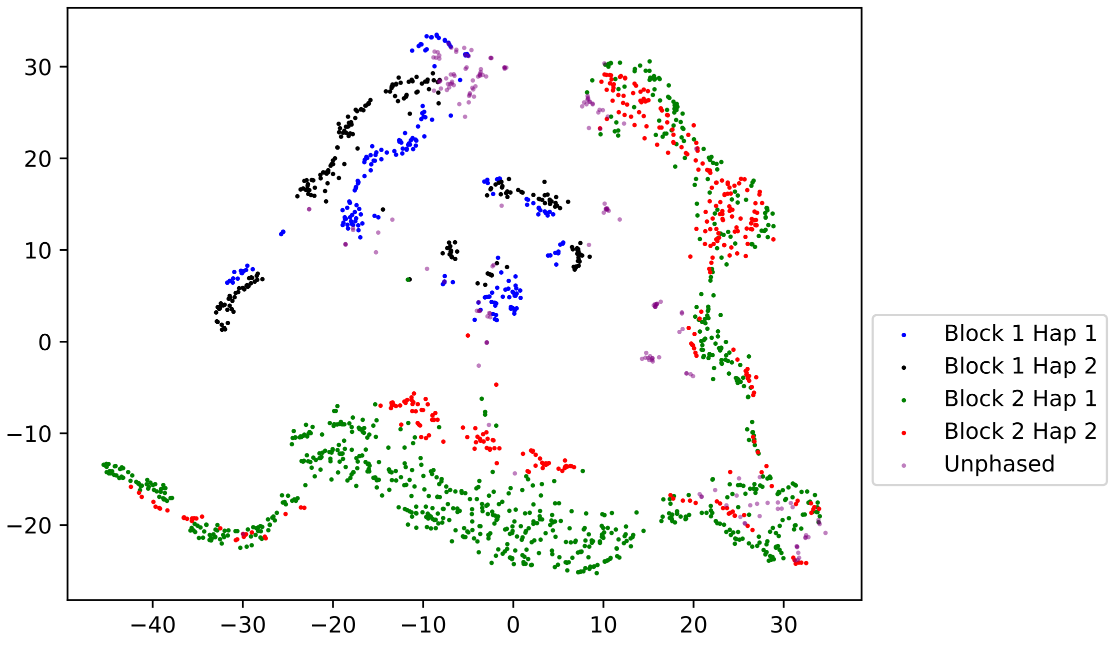

Figure 5.7: LSH with k=128, d=50

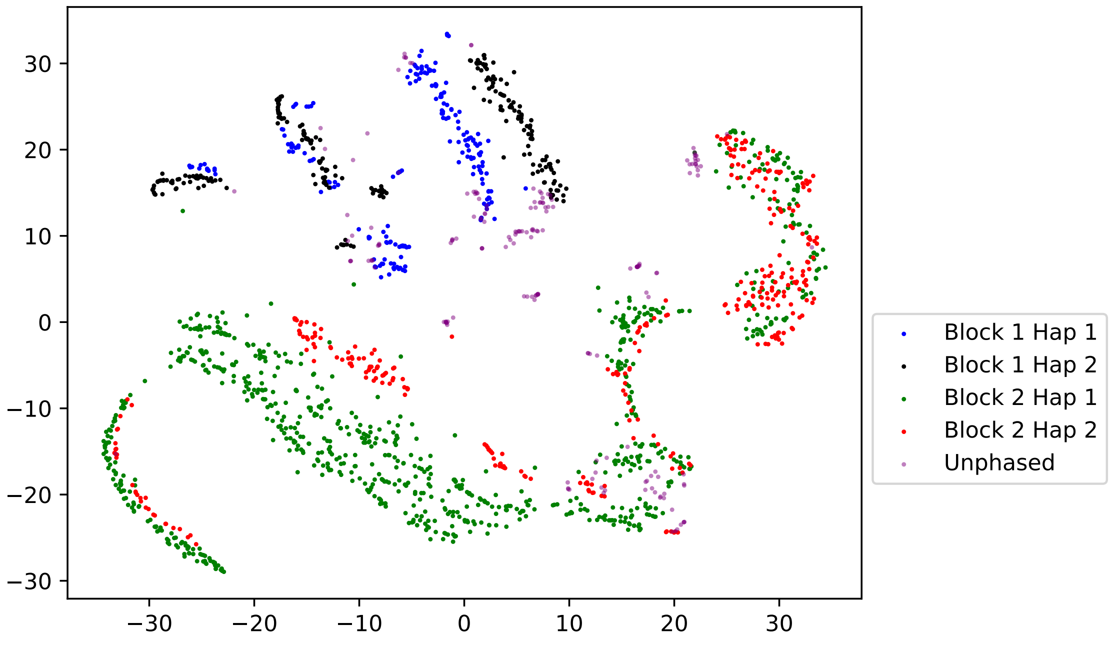

Figure 5.8: LSH with k=128, d=100

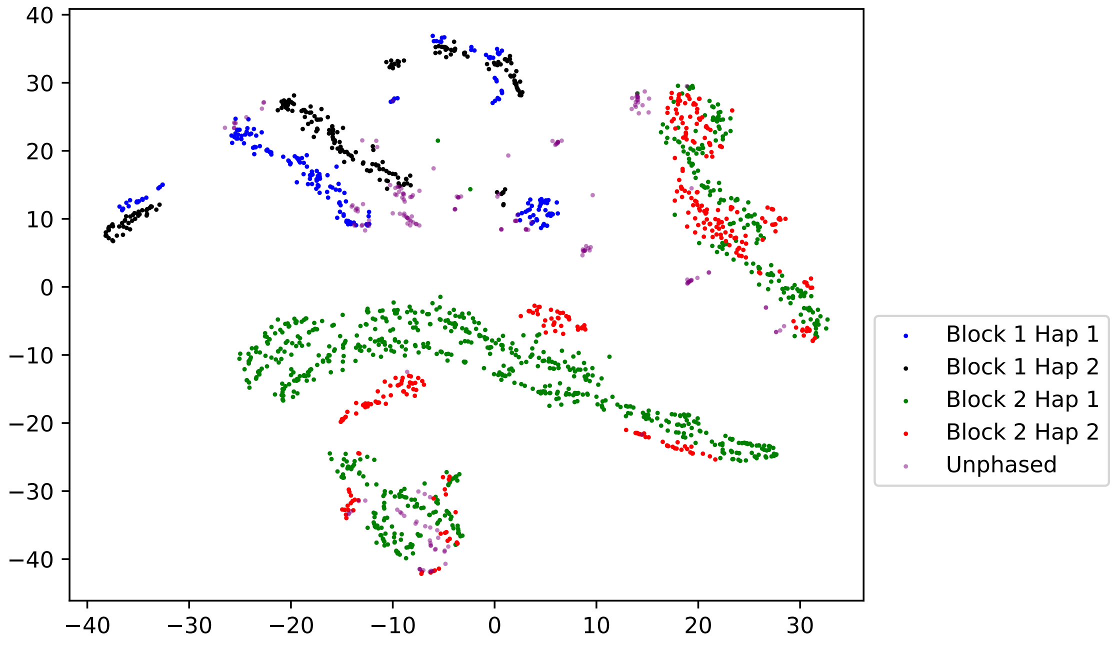

Figure 5.9: LSH with k=128, d=200

5.4 Other Phase Blocks in Chromosome

Figures 5.10 through 5.12 display other such results from the chromosome for different phase blocks, with $k = 15, d = 200$.

Where many phased reads are present in proportion to unphased reads, the model behaves as expected, and unphased reads are tucked between phased reads, indicating clear separations. Indeed, as an intuitive model might suggest, the unphased reads serve mostly as connections between clusters of candidate haplotypes.

Where the proportion of unphased reads to phased reads is large, the unphased reads seem to form a cluster of their own, representing the sequence-like nature of the reads due to their overlap. This spindly nature, we believe, arises from the fact that two near neighbors for unphased reads are positionally close, and therefore have large amounts of overlap.

With sparse phase blocks, one notices the high likelihood of unphased reads being nestled between phased reads, or serving as connectors between different phase blocks. Such sparse visualizations provide a more granular view, although such simple models may arise due to the small size of their training sets.

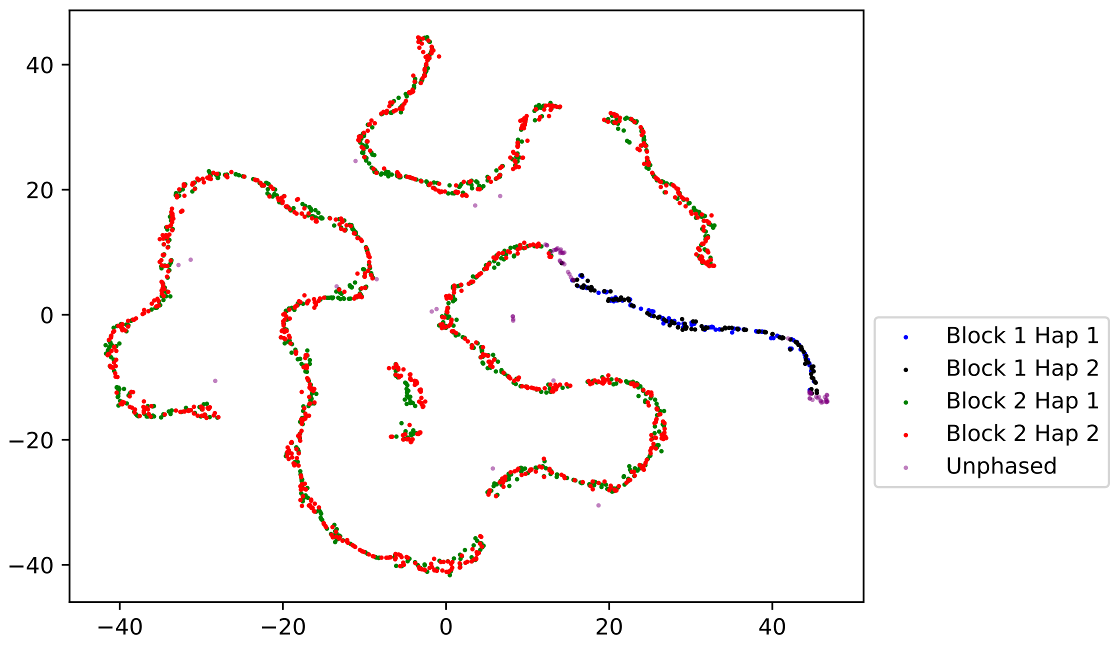

Figure 5.10: LSH with Greater Proportion of Phased to Unphased

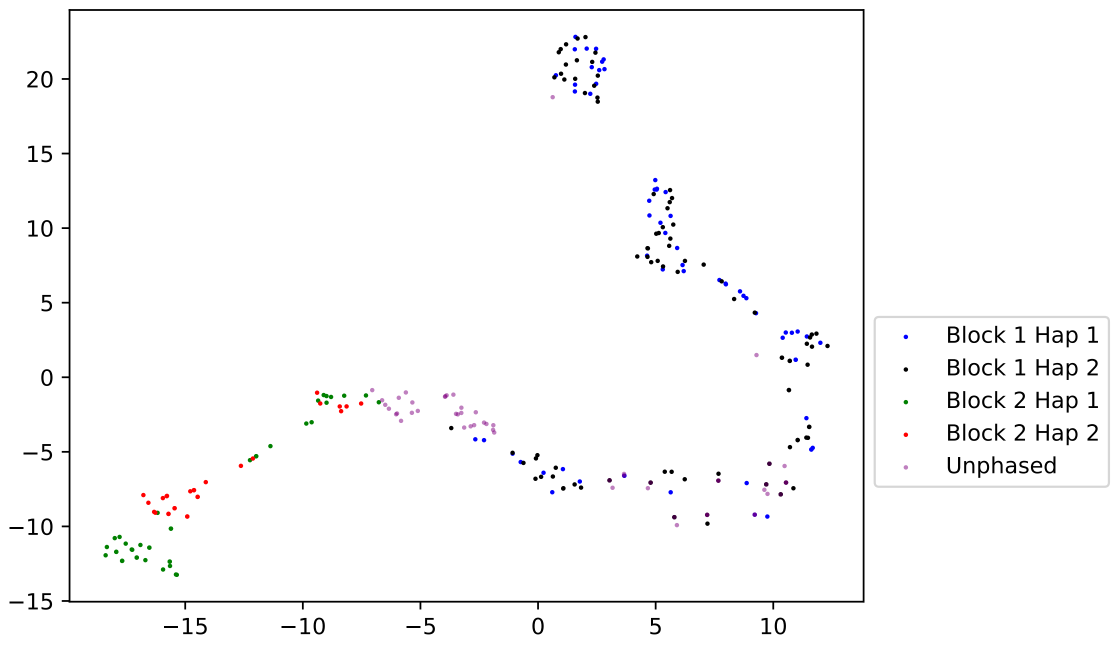

Figure 5.11: LSH with Sparse Phase Blocks

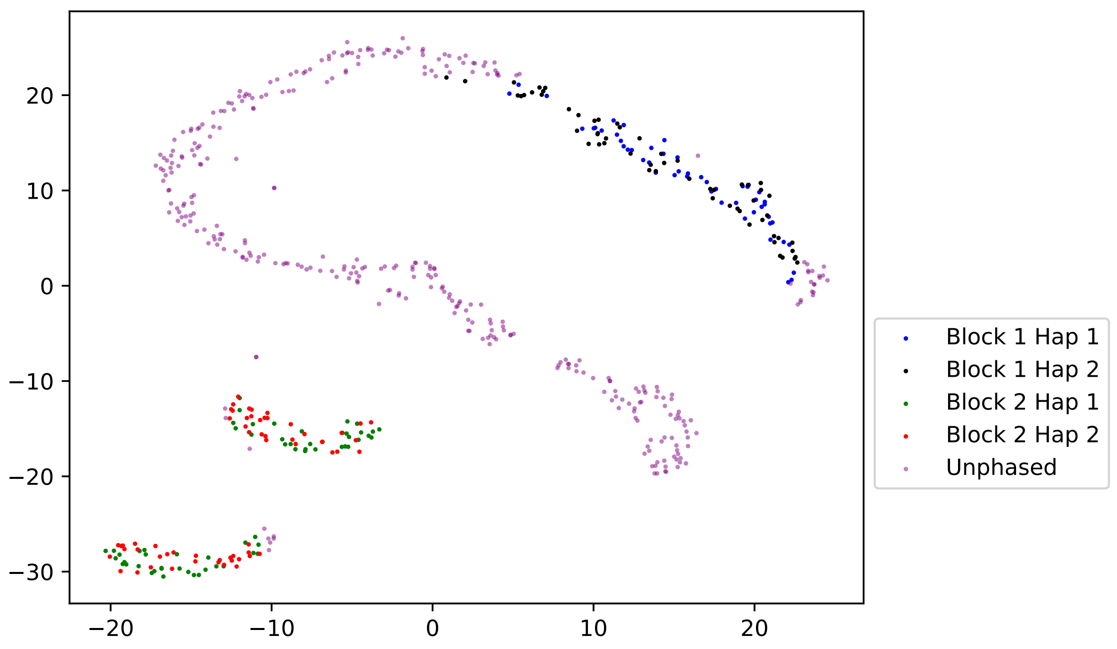

Figure 5.12: LSH with Greater Proportion of Unphased to Phased

5.5 Timing Comparison for LSH and Hex

Table 5.3 shows job completion time on ACCRE parametrized over k-mer size, embedding dimension, and hashing type.

One may note that hexadecimal hashing timing increases drastically with dimension and k-mer size increases, while LSH seems resistant. This is likely due to the fact that bucket identification size does not scale with k-mer size, and that due to this scaling in hexadecimal hashing, the training model must perform more disk access due to being unable to load the hashes from the hash file into memory.

Hashing	K-mer Size	Dimension	Timing
Hex	4	50	1:16
Hex	4	100	1:40
Hex	4	200	2:46
Hex	16	50	6:03
Hex	16	100	6:19
Hex	16	200	11:12
Hex	128	50	8:08
Hex	128	100	8:58
Hex	128	200	20:29
LSH	4	50	2:23
LSH	4	100	2:16
LSH	4	200	2:20
LSH	16	50	2:55
LSH	16	100	3:15
LSH	16	200	3:35
LSH	128	50	4:26
LSH	128	100	4:42
LSH	128	200	5:41

Table 5.3: Timing Comparison of Jobs
Timing is in min:sec format

5.6 Storage Comparison for LSH and Hex

Table 5.4 shows storage for various files on disk.

As one may note, model files and hashing files for LSH are much smaller as compared to hexadecimal hashing. This is to be expected, as one recalls that for hexadecimal hashing, due to character-wise preservation, hash sizes increase with k-mer size. Since the number of hashes is also not bounded in hexadecimal hashing, the number of vectors within the model is not bounded. Therefore, LSH stands as the clear winner in terms of storage.

Hashing	K-mer Size	Dimension	Hashing Size	Vector Model Size
Hex	4	50	72 MB	.060 MB
Hex	4	100	72 MB	.123 MB
Hex	4	200	72 MB	.251 MB
Hex	16	50	225 MB	398 MB
Hex	16	100	225 MB	806 MB
Hex	16	200	225 MB	1,633 MB
Hex	128	50	1,649 MB	2,341 MB
Hex	128	100	1,649 MB	4,517 MB
Hex	128	200	1,649 MB	8,885 MB
LSH	4	50	108 MB	.018 MB
LSH	4	100	102 MB	.051 MB
LSH	4	200	108 MB	.088 MB
LSH	16	50	100 MB	.452 MB
LSH	16	100	101 MB	.902 MB
LSH	16	200	101 MB	1.797 MB
LSH	128	50	99 MB	.460 MB
LSH	128	100	99 MB	.914 MB
LSH	128	200	99 MB	1.844 MB

Table 5.4: Storage Comparison of Jobs
MB in this context represents 1000 bytes, not 1024

5.7 Hex Visualizations

Figures 5.13 through 5.15 represent varying k-mers for hex hashing, with $d = 100$.

As may be seen, the results deteriorate as k-mer size increases. This likely arises due to insufficient training for each k-mer as compared to LSH; while in LSH, a k-mer reaps the benefits of the appearances of its fellow bucket members, hexadecimal hashing only updates a k-mer should it appear intact in the training set. Although the character n-gram model is meant to alleviate this, it appears insufficient for larger k-mers, where multiple appearances of the same k-mer are unlikely.

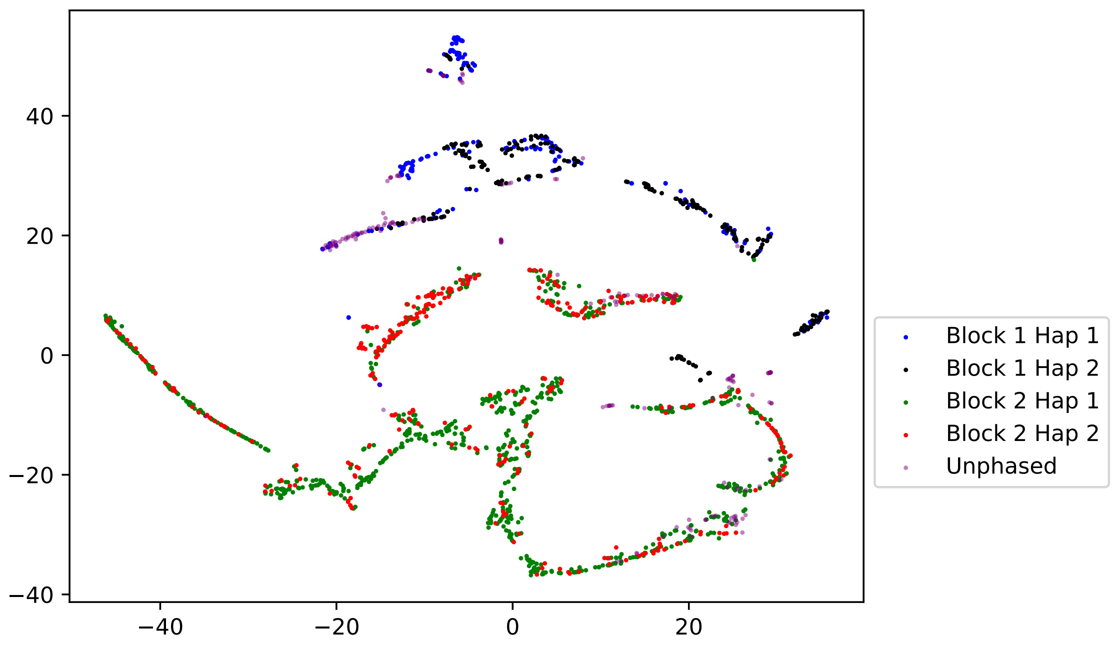

Figure 5.13: Hex with k=4, d=100

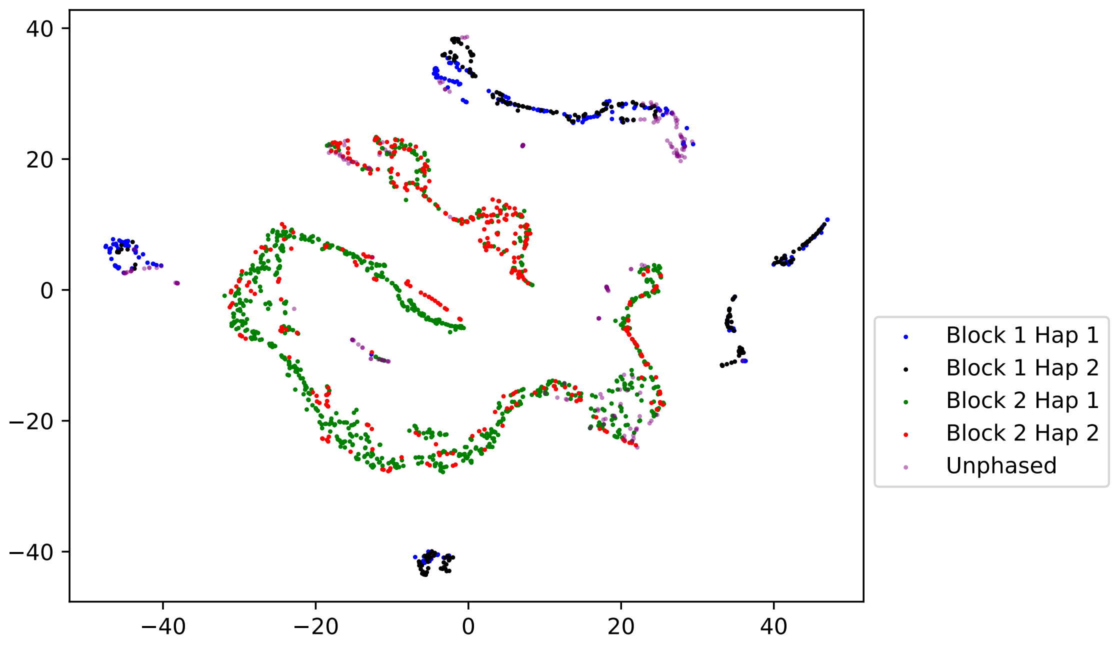

Figure 5.14: Hex with k=16, d=100

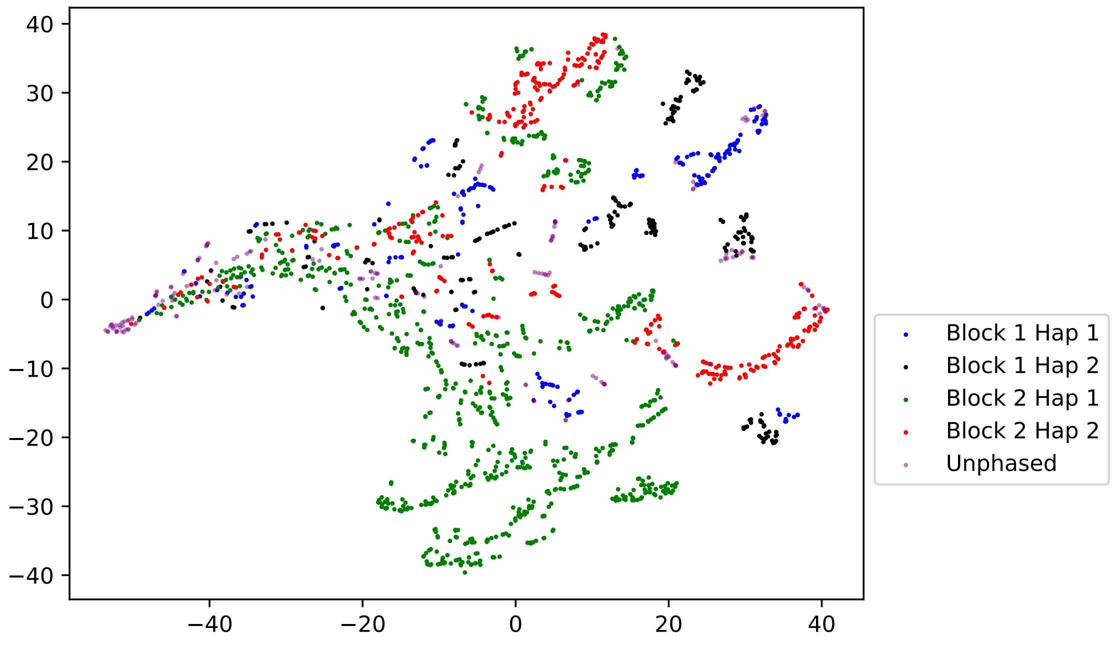

Figure 5.15: Hex with k=128, d=100

CHAPTER 6

Discussion

6.0.1 Training Parameters

For small datasets, the default values for the epoch and learning update rate suffice. However, for large amounts of genomic data, the epochs of training may be reduced to increase the speed of training, as a k-mer is encountered multiple times within a large corpus of k-mers. The justification behind this reduction of epochs was the sheer size of the corpus, where a k-mer could individually appear far more times than in a standard language model, depending on the k-mer length.

Due to this large corpus size assumption, average loss would remain convergent and low. Speed perhaps could be further improved by removing the character n-gram consideration, but their presence helps account for errors which may accumulate in rates as high as 15% in non-Hifi long read sequences (Rhoads and Au, 2015). The learning update rate, furthermore, may be disabled to ensure that for the reduced number of epochs, model learning does not diminish too rapidly.

6.1 Structure of Representation

Where ample data exists from both haplotypes and both phase blocks, the results demonstrate a fair shuffling, with the unphased reads being interspersed within a definite phase block. Where data from phased reads is sparse, however, insufficient training data leads to a difficulty in characterization.

Despite this, even in cases of insufficient training data, the structure of the data remains long and narrow, as opposed to scattered; this may reflect the relation between the sequences due to overlap of their subsequences. As data with t-SNE is usually scattered, perhaps the shape of the data might encode some important information.

6.2 Neighboring Measure

Due to distinctions in the shape of the data and the size of the phased data, the question arises over whether the closest neighbor is too simplistic of a measure to phase a sequence. Although the 20 (if available) nearest neighbors are stored for each phase block and haplotype, pure distance does not seem to be a reliable measure as one departs from the local region of a point.

Though the clusters preserve shape, they frequently move around on retraining, which might suggest that a local clustering algorithm is necessary as opposed to a single global measure of distance. To this purpose, an threshold-based clustering algorithm such as DBScan may be used (Pedregosa et al., 2011). This would preserve the separation between the clusters without requirement of specifying the number of clusters, given that a low threshold is selected.

6.3 Model Storage and Usage

LSH performed much better than hex hashing at reducing storage size for the model, which gives it a distinct advantage for scalability on larger, genomic data. Times for loading the hex trained model were also much long due to its size, which slows the embedding stage.

One possible solution is conversion to another format, although that would risk losing out on the character n-gram capabilities of FastText, which is undesirable as previously stated for error dampening measures. Another possible solution is to maintain the model as an object and query it directly prior to writing out its contents and exiting.

However, depending on the size, one might have to resort to swap files, which would threaten to undo the benefits gained in not reloading the model. A database that would be conducive to queries on such a model might be a potential answer for future work. Spark is one such framework for workflows that require scalability and parallelism (Zaharia et al., 2012). Its usage might improve timing and storage for both models, hex-based and LSH-based.

6.4 Classification

An astute reader might consider this problem, on a large scale, as a classification problem as compared to a word representation problem. Though this could be done locally, it is not possible to classify sequences globally, as the designation of haplotype number is preserved locally not globally. Haplotype 1 in one phase block, therefore, may be haplotype 2 in another phase block.

Still, on a local level, one might use classification on the phased sequences during the training stage so as to predict the labels for the unphased sequences. The timing and accuracy of this methodology in comparison to the current word representation methodology would be good candidates for future research.

6.5 Other Applications

A problem that might benefit from adoption of this module might be the barcode deconvolution problem, which was my prior focus. In this problem, lacking positional data, short reads belonging to the same barcode must be clustered so that a positional partition may be rendered unto the clusters within a barcode. As this module does not specify any number of clusters, nor uses positional data except in the first stage, the endeavor sounds promising.

6.6 GPU Capabilities

As FastText does not support GPU training, it might be beneficial to replace FastText with a model generated in PyTorch with CUDA capabilities (Paszke et al., 2019). Furthermore, with this replacement, the model within PyTorch could be customized to computational genomics, as opposed to its basis in NLP. As PyTorch has large amounts of inbuilt capabilities, this might even allow for models independent of FastText to arise.

CHAPTER 7

Conclusion

In this thesis, a natural language processing based module for the assignment of unphased reads was implemented. The components of this module are also modular, and may be changed as suits purpose. The module scalable to datasets of varying sizes, and does not assume the availability of large amounts of computational resources. Its adoption within an overarching structural variant detection pipeline led to better F1 scores as compared to alternatives such as Dipcall, PBSV, and CuteSV. Future work might apply different clustering algorithms, GPU based training, or parallelizable workflows to it.

References

(2015). Genome in a bottle—a human dna standard. *Nature*.

(2016). Na24385.

(2018). Pacbio structural variant (sv) calling and analysis tools.

Alkan, C., Coe, B. P., and Eichler, E. E. (2011). Genome structural variation discovery and genotyping.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Arxiv*.

Cheng, H., Concepcion, G. T., Feng, X., Zhang, H., and Li, H. (2021). Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature Methods*.

Edge, P. and Bansal, V. (2019). Longshot enables accurate variant calling in diploid genomes from single-molecule long read sequencing. *Nature*.

English, A. C., Menon, V. K., Gibbs, R., Metcalf, G. A., and Sedlazeck, F. J. (2022). Truvari: Refined structural variant comparison preserves allelic diversity. *Biorxiv*.

Gionis, A., Indyky, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing.

Heather, J. and Chain, B. (2016). The sequence of sequencers: The history of sequencing dna. *Genomics*.

Heller, D. and Vingron, M. (2019). Svim: structural variant identification using mapped long reads. *Bioinformatics*.

Heller, D. and Vingron, M. (2020). Svim-asm: structural variant detection from haploid and diploid genome assemblies. *Bioinformatics*.

Jiang, T., Liu, Y., Jiang, Y., Li, J., Gao, Y., Cui, Z., Liu, Y., Liu, B., and Wang, Y. (2020). Long-read-based human genomic structural variation detection with cutesv. *Genome Biology*.

Li, H. (2018). Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*.

Li, H., Bloom, J. M., Farjoun, Y., Fleharty, M., Gauthier, L., Neale, B., and MacArthur, D. (2018). A synthetic-diploid benchmark for accurate variant-calling evaluation. *Nature Methods*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Rhoads, A. and Au, K. F. (2015). Pacbio sequencing and its applications. *NCBI*.

Sedlazeck, F. J., Rescheneder, P., Smolka, M., Fang, H., Nattestad, M., von Haeseler, A., and Schatz, M. C. (2018). Accurate detection of complex structural variations using single-molecule sequencing. *Nature*.

Shi, L. and Chen, B. (2019). A vector representation of dna sequences using locality sensitive hashing. *Biorxiv*.

Ulyanov, D. (2016). Multicore-tsne. https://github.com/DmitryUlyanov/Multicore-TSNE.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*.

Wenger, A. M., Peluso, P., Rowell, W. J., Chang, P.-C., Hall, R. J., et al. (2016). Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature Biotechnology*.

Zaharia, M., Chowdhury, M., Das, T., et al. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing.