School of Computer Science
University of Technology Sydney

Unsupervised learning for high performance compression of genomic data collections

A thesis submitted in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

by

Tao Tang

June 2021

CERTIFICATE OF ORIGINAL AUTHORSHIP

I, Tao Tang declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Computer Science at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature of Candidate

Production Note: Signature removed prior to publication

Date 11/03/2021

Acknowledgments

First and foremost, I offer my deepest gratitude to my supervisor, Professor Jinyan Li, whose expertise was invaluable in formulating the research questions and methodology, for his extensive support and guidance during last three years of my PhD studies. His guidance including scientific writing and research ideas made all of this work possible. His insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I am very thankful to Dr. Yuansheng Liu, who is our team member. Dr. Yuansheng Liu has vast knowledge in the area of bioinformatics, especially the compression of NGS data, he gives many insightful suggestions and comments to improve the quality of our projects. I would also like to thank other team members: Dr. Chaowang Lan, Dr. Hui Peng, Xiaocai Zhang, Xuan Zhang, Dr. Yi Zheng and Dr. Zhixun Zhao, for their kind help during my studies. The fantastic activities with them are enjoyable and relaxing under the heavy pressure of research.

This work would not be possible without the people with whom I collaborated and whose help was essential in many projects I participated in: Professor Gyorgy Hutvagner, Dr. Kun Yu.

Furthermore, I gratefully acknowledge all the organizations provided the scholarships including International Research Scholarships, ARC Discovery Scholarship.

I am deeply grateful to my wonderful parents for their support and love throughout all my life. I also extend my deep gratefulness to my relatives and friends in China.

Tao Tang
UTS, Sydney, Australia

Feb 2021

Contents

List of Figures

List of Tables

List of Publications

Below is the list of journal papers associated with my PhD research:

Journal Papers Published

- **Tang, T.**, Liu, Y., Zhang, B., Su, B., & Li, J. (2019). Sketch distance-based clustering of chromosomes for large genome database compression. *BMC genomics*, 20(10), 1-9.
- **Tang, T.**, & Li, J. (2021). Transformation of FASTA files into feature vectors for unsupervised compression of short reads databases. *Journal of Bioinformatics and Computational Biology*, 2050048-2050048.
- **Tang, T.**, Hutvagner. G., Wang, W. & Li, J. Assembly-improved compression of genomic short reads via error correction: survey and advancement. Under review.

Abstract

The advanced next-generation sequencing (NGS) technologies have launched a new era of all fields of genetics. However, the vast quantity of data generated by NGS technologies also proposed great challenges to data storage, transmission and analysis. In this thesis, we focus on the compression of multiple data collections of short reads and assembled genome, we also explore the relationship between compression, error correction and de novo assembly of short reads data. First, we introduce an efficient clustering-based reference selection algorithm for the compression of genome databases. This method clusters the genomes into subsets of highly similar genomes using MinHash sketch distance, then applies a two-level compression based on the clustering result. The compression ratio gain of our approach can reach up to 20-30% in most cases for the datasets from NCBI, the 1000 Human Genomes Project and the 3000 Rice Genomes Project.

Furthermore, we propose a new clustering-based method for the compression of short reads datasets. Our approach transforms each file into a feature vector for clustering, then compresses the files in the same group together to increase the total number of detected overlappings during compression. The experiments show that our method achieves 20%-30% improvements in compression ratio than the previous one-by-one compression.

Finally, we review the relationship between reference-free compression, MSA based error correction and de novo assembly of short reads data. We demonstrate that high quality error correction can significantly reduce the number of mismatched nucleotides during reference-free compression and

hence improve the final compression ratio. The experiment results verify our estimation and show that the same error correction also has a positive effect on de novo assembly in most cases. In addition, we also propose a path graph based method for compression of short reads datasets.

Chapter 1

Introduction

Sequencing is the process of determining the order of nucleotides in DNA molecules, which is the main driving force in biologic and medical research (Zhu, Zhang, Ji, He & Yang 2015, Mardis 2011, Koboldt, Steinberg, Larson, Wilson & Mardis 2013). The bases in DNA molecules include adenine(A), guanine(G), cytosine(C) and thymine(T).

The release of the first high-throughput sequencing technology in the mid-2000s (Margulies, Egholm, Altman, Attiya, Bader, Bemben, Berka, Braverman, Chen, Chen et al. 2005, Shendure & Ji 2008), also referred as Next Generation Seqeuncing (NGS), has heralded a 50,000-fold drop in the cost of human genome sequencing for Human Genome Project (Goodwin, McPherson & McCombie 2016), which undeniably reinvigorates the field of bioinformatics research (Shendure & Ji 2008). In the past 15 years, NGS technologies have gradually replaced Sanger Sequencing (Sanger, Nicklen & Coulson 1977) to be the mainstream sequencing technology due to its lower cost with continually decrease (Zhu, Zhang, Ji, He & Yang 2013): in 2001, it takes around US$5000 to sequence one megabase of human genome DNA data, until the end of 2015, the cost of human genome DNA sequence data per megabase is US$0.3.

The advanced NGS technologies are not only developed with continually decreasing cost but also increasing throughput, which have provoked a

number of nation-wide and international genomic discovery projects in biomedical (Lawrence, Stojanov, Mermel, Robinson, Garraway, Golub, Meyerson, Gabriel, Lander & Getz 2014), genotyping (Danek, Deorowicz & Grabowski 2014), and metagenomics assembly (Pell, Hintze, Canino-Koning, Howe, Tiedje & Brown 2012) research, such as the 1000 genomes project (Consortium et al. 2012), giant salamander genome sequencing (Yan, Lü, Zhang, Yuan, Zhao, Huang, Wei, Mi, Zou, Xu et al. 2018), 3000 rice genomes project (Wang, Mauleon, Hu, Chebotarov, Tai, Wu, Li, Zheng, Fuentes, Zhang et al. 2018), ENCODE Project (ENCODE Project Consortium 2012), and the 100,000 Genomes Project (England 2016).

The growth rate of NGS data generated by large scale projects has outpaced the one predicted by Moore's law, which results in petabytes of sequence or reads data (e.g. NCBI Sequence Read Archive maintained over 14 petabytes NGS data as of Dec 2020). One of the biggest challenges proposed by the vast quantity of NGS data is the economic storage and transmission. Efficient compression is a potential way to solve that. The general-purpose compression algorithms like gzip (http://www.gzip.org/) and bzip2 (http://www.bzip.org) are frequently used for storage of genomic data (Zhu et al. 2015). However, these algorithms did not efficiently utilize the intrinsic features of NGS data such as repetitive subsequences, small alphabet size (Deorowicz & Grabowski 2013, Wandelt, Bux & Leser 2014), and especially the redundancy (overlapping) between reads. After the storage and transferring of large scale NGS data become routine, several specialized compression algorithms for NGS data have been developed and achieved much higher compression ratio than general-purpose compression algorithms. However, for compression of multiple datasets (files), the previous method normally selects an external reference manually (for reference-based compression) or compress files separately (for reference-free compression), which limits the performance of large scale NGS data compression.

Though NGS data have already been distinguished by labels such as species, read length, DNA/RNA sequencing data, the division based on

these is not accurate enough for dedicated compression. As existing labels from human supervision are insufficient for the compression of NGS data (including reads and genome data), groups the datasets with unsupervised learning (Hastie, Tibshirani & Friedman 2009) is an optional solution.

The relationship between compression, error correction and de novo assembly of NGS data is another fundamental problem, especially whether compression can benefit from error-reduced input data.

The main objective of this thesis is the efficient compression of multiple large-scale NGS data sets via unsupervised learning, and exploration of relationship between data compression, error correction and de novo assembly of short reads data.

1.1 Compression of Genome Sequence Data

Whole genome sequencing (WGS) is considered to be one of the most comprehensive genetic tests in bioinformatics study (Lupski, Reid, Gonzaga-Jauregui, Rio Deiros, Chen, Nazareth, Bainbridge, Dinh, Jing, Wheeler et al. 2010). The assembled whole genome sequencing data is normally stored in FASTA (Pearson & Lipman 1988) format file, which is a text-based format for biological sequence comparison. It begins with an identifier and multiple '>' (or ';' in a few cases), followed by multiple lines of sequence data. The sequence data is constructed by the standard IUB/IUPAC nucleic acids in base pairs represented using single letter codes, an example of FASTA format data is shown in Fig 1.1.

The size of whole genome data varies from different life forms and species, e.g. $10^6 - 10^7$ base pairs for gram positive bacteria, $10^8 - 10^{11}$ for flowering plants and $10^{9.5}$ for mammals. The total length of a human reference genome is around 3 billion base pairs, which requires roughly 3 GB of memory (Ochoa, Hernaez & Weissman 2015). With the advanced sequencing technologies, more and more genomes are sequenced, the increasing size of NGS datasets has produced great interests in compression of genome data.

```
>gi|528476558|ref|NC_018928.2| Homo sapiens chromosome 17
TCCACTCACTGAGACAATAGACCCAAGCACATCAGCTCTGAGGCCTC
AGCACATCAGCTCTGAGGCCTCCACTCACTGAGACAATAGACCCAAG
TCACTGAGACAATAGACCCAAGCACATCAGCTCTGAGGCCTCCACTC
AACAGCTCTGAGGGCTCCACTCACTGAGACAACAGACCCAAGAACAA
>gi|528476670|ref|NC_018912.2| Homo sapiens chromosome 1
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
CCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAA
ACCCTAACCCTAACCCTAACCCTAACCCAACCCTAACCCTAACCCTAACCCTAACCCTA
TAACCCTAACCCTAAACCCTAAACCCTAACCCTAACCCTAACCCTAACCCTAACCCCAA
```

Figure 1.1: An example of FASTA format data

Before NGS technologies, the traditional genomic compression focused on relatively small genomic data produced by Sanger sequencing (Sanger et al. 1977). The traditional genomic compression focused on searching complemented palindromes and approximate repeats among genomic data. These algorithms can be classified into two types: substitutional based and statistical based. The substitutional based compression methods record highly repetitive subsequence and replace each subsequence in original genome with encoded ones, BioCompress (Grumbach & Tahi 1993) and BioCompress2 (Grumbach & Tahi 1994) utilize Lempel-Ziv style algorithms (Ziv & Lempel 1977) to compress the repeats and palindromes, GenCompress (Chen, Kwong & Li 2001) and DNACompress (Chen, Li, Ma & Tromp 2002) identify both exact and approximate repeats, GeNML (Korodi, Rissanen, Astola et al. 2007) encodes approximate repeats using normalized maximum likelihood (NML) model. Statistical compression methods such as XM (Cao, Dix, Allison & Mears 2007) construct a prediction model (e.g. high order Markov chain) to compute the probability distribution of each base, then apply encoding based on it. CTW-LZ (Matsumoto, Sadakane & Imai 2000) can be considered as a hybird approach, which applies both LZ scheme (Ziv & Lempel 1978) and context tree weighting (CTW) prediction model (Willems, Shtarkov & Tjalkens 1995) for the compression.

With the continuing exponential accumulated genomic data produced

by NGS technologies, traditional compression tools are impractical due to memory usage and computation time. More and more recent compression methods have been proposed, which mainly have two types: reference-free and reference-based compression.

The reference-free methods such as BIND (Bose, Mohammed, Dutta & Mande 2012) and DELIMINATE (Mohammed, Dutta, Bose, Chadaram & Mande 2012) focus on the encoding strategy of bases (A,C,G,T) to achieve higher compression efficiency on NGS data than the traditional methods.

As advanced sequencing technologies produced multiple genomic sequences of each species, reference-based compression becomes a preferred option when the similar genomic data is available. As its name indicates, reference-based compression requires a reference sequence for the target sequence. For target sequence X and a given reference sequence Y, the methods search common substring between X and Y, then encode the common substring in X as the corresponding position and length in Y. Since Brandon et al. attempt to encode the difference between target and reference sequence (Brandon, Wallace & Baldi 2009), several reference-based compression methods have been proposed and achieved hundreds of folds compression ratio on human genome data (Pavlichin, Weissman & Yona 2013, Deorowicz, Danek & Grabowski 2013).

The advantage of reference-based compression is not only supported by its performance on genomic data, but also mathematical theory: according to Shannon's source coding theorem (Shannon 1948), the mathematical limit on how well X can be losslessly compressed via reference-free methods is expressed as $H(X)$, which is the entropy of X, for reference-based compression, the mathematical limit equals $H(X|Y) = H(X) - I(X,Y)$, where $I(X,Y)$ denotes the amount of information obtained about X when Y is observed or vice versa. As $I(X,Y) \geq 0$, it is clear that the mathematical limit of reference-based compression is smaller than reference-free compression, and the difference between mathematical limitation is $I(X,Y)$, which can be considered as the similarity between X and Y.

Although there is normally a high similarity between target genomes and the reference of the same species, the compression ratio between target and different reference sequences of still have a high variance. For instance, the compression ratio for genome hg19 by GDC2 using seven different reference genomes varied remarkably from 51.90 to 707.77 folds. The compression ratio of other state-of-the-art methods with different reference genomes also have high variance in experiment (Liu, Peng, Wong & Li 2017). Therefore, grouping similar genomes and species reference identification within the clusters are of great significance in the compression of large scale genome databases.

In this thesis, we focus on the reference selection for multiple target sequences via unsupervised learning.

1.2 Compression of Reads Data

The raw sequencing data generated by NGS is normally referred to as reads data and stored in FASTQ format (Cock, Fields, Goto, Heuer & Rice 2009). FASTQ format is also a text-based format that represents the biological sequence data and quality score with **ASCII** character. A FASTQ file can contain millions to billions of entries, each one consists of four parts:

1. A '@' character followed by sequence identifier, sometimes a description is also included.

2. A line of reads data ('A','C','G','T','N').

3. A '+' symbol, optionally followed by the same sequence identifier as in above.

4. A line of quality scores with the same number of read length, each one represents the probability that the corresponding base is called correct.

An example of a FASTQ file with three sequences is depicted in Fig 1.2.

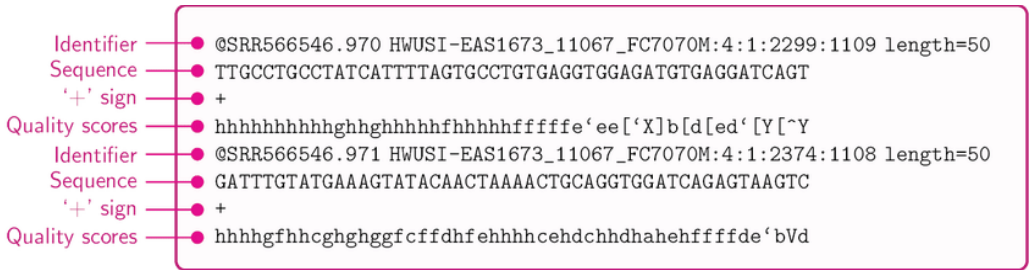


Figure 1.2: An example of FASTQ format data

The compression of raw sequencing data includes three parts, the sequence identifier, reads data, and quality scores. Due to the different characteristics of these three parts of data, most compression tools for NGS data compress them independently or only compress one of them. As the quality scores and reads data contain more information, existing methods normally focus on optimizing the compression of these two parts.

For the compression of quality score, lossy compression is more preferred due to the larger alphabets and higher entropy. Most of the compression tools smooth the quality scores of the bases predicted to be correct (Yu, Yorukoglu, Peng & Berger 2015, Ochoa, Hernaez, Goldfeder, Weissman & Ashley 2016, Shibuya & Comin 2019), it has been proved that strategy can achieve high compression ratio without affecting the downstream analysis such as SNP calling (Cánovas, Moffat & Turpin 2014).

As a result of high-throughput sequencing technologies, the reads data contain complicated redundancy such as standard/reverse-complementary duplicate and overlap. Hence the key idea of compression of reads data is utilizing the inter-similarity between the reads data in the same set, which is referred as **reference-free compression**. A typical reference-free compression method for reads data consists of three steps: Firstly, the redundancy in reads data are searched by heuristic method. Then reads with redundancy are merged into longer contigs. At last, the reads are encoded with regard to contigs.

Reference-free compression algorithms such as HARC (Chandak, Tatwawadi & Weissman 2018), minicom (Liu, Yu, Dinger & Li 2019), PgRC (Kowalski & Grabowski 2020*b*) have achieved a balance between practicability and performance on single large scale reads set. However, the utilization of redundancy between reads sets is still lacking. Current compression algorithms tend to compression one single reads set each time, the intra-similarity between different reads sets is normally ignored.

As it is have been previously proved that two reads are likely to have a large overlap if they share some common features (Roberts, Hayes, Hunt, Mount & Yorke 2004). We estimate that detecting reads sets with higher similarity and then compress these reads together will significantly increase the overall compression ratio.

In this work, we focus on improving the compression ratio of multiple reads datasets via unsupervised learning.

1.3 Effect of Error Correction on Compression and De Novo Assembly of NGS Data

High-throughput NGS technologies can generate hundreds of millions of reads data in one single experiment (Cleary & Witten 1984). One of the most essential downstream analysis of the massive amount of reads data is de novo assembly, which is the process of merging reads data into longer contigs without using reference genome.

For the assembly short reads data generated by NGS technologies, de Bruijn graph (Zerbino & Birney 2008) is widely used: In this data representation, each unique k-mer in reads set is considered as a node and the nodes with $k-1$ overlaps are connected with edges, longer contigs is then extracted from the path of constructed graph. De Bruijn based assemblers show high efficiency on short reads data, however, it also suffers from the errors in NGS data (Heydari, Miclotte, Demeester, Van de Peer & Fostier 2017).

Compare to Sanger sequencing, NGS technologies have a relatively high error rate, the errors are mainly substitution errors and the rate ranges from 0.26% to 12.86% on different platforms (Quail, Smith, Coupland, Otto, Harris, Connor, Bertoni, Swerdlow & Gu 2012). Due to the characteristics of de Bruijn graph, a single substitution error can lead up to k erroneous nodes and corresponding edges. Most assembly tools (Zerbino & Birney 2008, Allam, Kalnis & Solovyev 2015, Koren, Walenz, Berlin, Miller, Bergman & Phillippy 2017) have built-in procedure to correct the errors based on the detection of erroneous structures in constructed graph.

On the other hand, the substitution errors in NGS data also affect the performance of reference-free compression: As mentioned before, the compression tools merge reads into contigs and then encode reads with regard into contigs, when single substitution error occurs in reads, the original common subsequence between contigs and these reads will change to two common subsequences partitioned by a base, which requires much more space to encode than the original data.

Several methods have been proposed to correct the errors in NGS data. Multiple Sequence Alignment (MSA) based error correction methods select a subset of reads for each read r, then conduct alignment on r and the subset of reads to detect the erroneous bases. State-of-the-art MSA based error correction tools such as Karect (Allam et al. 2015) achieved over 99% precision in experiments.

It can be seen that the de novo assembly, compression and error correction of reads data have the same characteristic: most of these methods utilize the high coverage of NGS data and the similarity between reads. However, the research on the relationship between them is lacking.

In this work, we explore the effect of MSA based error correction on reference-free compression and de novo assembly.

1.4 Research Contributions

In this thesis, we focus on three research problems encountered by the vast quantity of NGS data:

1. The compression of multiple genome sequences.
2. The compression of multiple reads datasets.
3. The effect of error correction on compression of reads data.

By dividing the datasets into subgroups with higher similarity via unsupervised learning and applying compression based on that, we significantly improve the compression ratio of genomics and reads data with relatively short additional computation time. We also demonstrate the relationship between reference-free compression, de Novo assembly and MSA based error correction of short reads data.

Our contributions can be listed as the following:

- We introduced ECC (Tang, Liu, Zhang, Su & Li 2019), an efficient clustering-based reference selection algorithm for the compression of genome databases. This algorithm uses MinHash technique to measure the difference between each reads dataset and constructs a distance matrix based on that. Then it applies subtractive clustering to determine the number of subgroups in datasets and K-medoids clustering to determine the reference of each sequence. In the compression stage, we design a two level compression structure based on the clustering result: within each subgroup, all the genomes are compressed with the centroid genome as reference except the centroid genome itself. Then all the centroid genomes are considered as one cluster and a final genome is selected to be the reference for compressing other centroid genomes.

  By comparing the performance of ECC with traditional single reference compression on six state-of-the-art reference-based compression algorithms

and three collections of datasets, ECC can reach up to 20-30% compression ratio gain on the datasets of 171, 1012 and 3128 GB. The best compression ratio of 171Gb data increases from 351.74 folds to 443.51 folds.

- We proposed MRC (Tang & Li 2020), a novel clustering method for compressing reads datasets. MRC transforms each reads dataset into a feature vector and applies a variant of K-means clustering. Then it compresses the reads dataset in the same group together. In experiments, MRC outperforms the previous one-by-one compression in compressing four collections of reads sets with four state-of-the-art compression algorithms, the compression gain ranges from 4% to 12% for the most efficient compression algorithms, the additional computation time is less than 10% of the compression time.

- In our work, we review the common characteristics of de novo assembly, compression and error correction. We found that high quality MSA based error correction can significantly reduce the number of mismatched nucleotides in reference-free compression, and hence improve the final compression ratio. We also found that MSA based error correction outperforms the built-in error correction procedure of the assembly tools in most test cases.

We apply our methods and the previous compression tools in the same area on the same datasets, then compare the compression ratio and speed performance to evaluate the effect of our methods. The experiments results show that our approach outperform the previous tools in most cases. We also evaluate the accuracy of a part of clustering results via other methods such as Principal Component analysis.

1.5 Thesis Organization

This thesis is composed of six chapters and is structured as following. In **Chapter 1**, we introduce our research problems and the background, the objective and contribution made by this thesis. In **Chapter 2**, we review the state-of-the-art algorithms related to our research problems, including reference-based compression of genomic data, reference-free compression of reads data, de novo assembly and error correction of short reads data. Then **Chapter 3** and **Chapter 4** present our new methods ECC and MRC respectively. ECC is a clustering-based approach for compression of multiple genomic sequences and MRC is for selecting to-be-compressed-together reads sets for compression of multiple reads sets. **Chapter 5** presents our survey on the relationship between compression, error correction and de novo assembly, especially the improvements on reference-free compression brought by error-reduced input data. **Chapter 6** provides a summary of this thesis and potential directions for my future research.

Chapter 2

Related Work and Literature Review

In this chapter, we present the previous work that relates to our thesis. We first review the previous methods for compression of genomic data in Section 2.1. Then we review the previous methods for compression of short reads data in Section 2.2. Finally, we survey the existing methods for de novo assembly and error correction of short reads data in Section 2.3.

2.1 Reference-based Genome Compression

As mentioned before, reference-based compression is a more preferred option for genomic data. The key idea of the existing reference-based genome compression algorithms is to map subsequences in the target genome sequence to the reference genome sequence (Zhu et al. 2013). Firstly, an index such as a hash table or a suffix array is constructed from the reference genome to reduce the time complexity of the search process. Then an encoding strategy such as LZ77 (Ziv & Lempel 1977) is applied to parse the target sequence to position number and length of the subsequence with regard to the reference sequence or mismatched subsequence. For instance, a subsequence in the target sequence is encoded as "102 72", which stands for that this

subsequence is identical to the subsequence from position 102 to 173 in the reference genome.

For a set of target genome sequences, the similarity between the reference sequence and the selected target sequence has a large effect on compression ratio. And some specialized procedures are designed for the compression of multiple genome sequences.

Existing attempts for reference selection in the compression of genome sequence databases can be categorized into three types. The first category selects a single reference genome to perform one-by-one sequential reference-based compression on all target genomes, which is named straightforward reference-fixed approach as in the previous section. Most of the reference-based compression algorithms applied that on genome set compression and select the single reference sequence randomly from the genome database, such as HiRGC (Liu et al. 2017), GECO (Pratas, Pinho & Ferreira 2016), ERGC (Saha & Rajasekaran 2015), iDoComp (Ochoa, Hernaez & Weissman 2014), CoGI (Xie, Zhou & Guan 2015), RLZ-opt (Kuruppu, Puglisi & Zobel 2011), RLZAP (Cox, Farruggia, Gagie, Puglisi & Sirén 2016). GDC (Deorowicz & Grabowski 2011) and FRESCO(Wandelt & Leser 2013) select one single reference with a heuristic technique and provide fast random access. MRSCI (Wandelt & Leser 2015) proposes a compression strategy that splits string set into references set and to-be-compressed set and then applies a multi-level reference-based compression.

The second category of algorithms utilizes not only one fixed reference for the compression of all sequences, but also the inter-similarity of the whole sequence set. Then it parses the subsequences not only based on the initial references but also the recorded pair. In other words, it considers all the compressed sequences as a 'potential reference' for the current compression. GDC2 (Deorowicz, Danek & Niemiec 2015) applies a two-level Ziv Lempel factorization (Ziv & Lempel 1977) to compress large set of genome sequences. MSC (Cheng, Wu, Law & Siu 2015) utilizes both intra-sequence and inter-sequence similarities for compression via searching subsequence matches

in reference sequence and other parts of the target sequence itself, the compression order is determined by a recursive full search algorithm.

The third category of algorithms selects reference via unsupervised learning. RCC (Cheng, Law & Siu 2017) performs clustering on the local histogram of dataset and derives a representative sequence of each cluster as the reference sequence for the corresponding cluster. A final representative sequence is then selected from the representative sequence set. For each cluster, the sequence data is compressed based on intra-similarity and inter-similarity with reference to the corresponding representative sequence.

The performance of the first and second category of methods depends on the reference sequence that selected manually, which requires prior knowledge about sequence similarity to obtain a high compression ratio. The third category of methods such as RCC (Cheng et al. 2017) enables reference selection via unsupervised learning, however, the derivation of representative sequence in RCC requires a large amount of time for assembly.The computation time of RCC is $O(N^2L + L^2)$, where N is the number of sequences and L is the average length of sequences. hence not suitable for large scale databases in real experiments. In further experiments, the average compression time of RCC on single human genome is longer than 4 hours and could not work on large human or rice genome sequence set.

2.2 Short Reads Compression

Short reads data refer to the NGS data with fixed and small read length (36-200bp). The compression of short reads data can be classified into two types: reference-based compression and reference-free (de novo) compression.

Reference-based compression methods align the subsequqnce in target reads sets to reference reads set and record the information for further decompression. Reference-based algorithms such as pathenc (Kingsford & Patro 2015), LWFQZip (Zhang, Li, Yang, Yang, He & Zhu 2015) have demonstrated superior performance on reads data, however, due to

the uncertain availability of proper reference data and time complexity of sequence alignment, the practicability of reference-based compression is restricted.

Compared to reference-based compression, reference-free compression shows higher practicability and hence more preferred for short reads data. The key idea of reference-free compression is to utilize the redundancy information between the reads in the same set to form groups of reads and de-novo assemble them into consensus sequences (contigs), and then encode the reads with regard to these consensus sequences, some general compression tools such as gzip, 7zip are used for final encoding. Existing reference-free compression algorithms include reference-free algorithms such as ReCoil (Yanovsky 2011), Quip (Jones, Ruzzo, Peng & Katze 2012), Fqzcomp (Bonfield & Mahoney 2013) and Assembltrie (Ginart, Hui, Zhu, Numanagić, Courtade, Sahinalp & David 2018).

SCALCE (Hach, Numanagić, Alkan & Sahinalp 2012) groups the reads based on the longest core substring, then sorts reads based on the lexicographical order with respect to the position of the core substring.

ORCOM (Grabowski, Deorowicz & Roguski 2014) groups reads via canonical minimizers (called signatures in this method), then reorder the reads to move overlapping reads possibly close to each other, in the last phase, it produces several (interleaving) streams of data, then encode these data by PPMD (Shkarin 2002) or variant of arithmetic coder (Salomon & Motta 2010).

Mince (Patro & Kingsford 2015) implements a data-dependent bucketing scheme to bucket similar reads together and reorder the reads in each bucket, it also handles paired-end reads by concatenating the left and right ends of the pair together in accordance with a user-provided library type.

HARC (Chandak et al. 2018) searches the maximum overlaps of the reads in the set to locate their approximate positions in a genome and then creates contigs from reads in the neighborhood of these positions in the genome. SPRING (Chandak, Tatwawadi, Ochoa, Hernaez & Weissman 2019) is an

advanced version from the same team of authors, improving HARC with options such as compressing reads with arbitrarily long reads lengths.

FaStore (Roguski, Ochoa, Hernaez & Deorowicz 2018) and Minicom (Liu et al. 2019) both group reads based on the redundancy information contained in sub-string patterns called k-minimizers (Roberts et al. 2004); the reads in these groups are then de-novo assembled or condensed into contigs for referential encoding.

A more recently published method PgRC (Kowalski & Grabowski 2020*b*) divides the whole set of reads into two groups: high-quality (without 'N' base) and low-quality (with 'N' bases), then generates an approximate shortest common superstring over each group via merging reads with suffix-prefix overlapping to the existing string until no reads are left, which is referred as pseudo genome construction (Kowalski, Grabowski & Deorowicz 2015). PgRC 1.2 (Kowalski & Grabowski 2020*a*) is a multi-thread version of PgRC with improvements in compression speed.

Overall, reference compression normally consist of three stages: (1) reorder the reads or divide reads into subgroups to reduce computation time in further stage (2) search overlapping between reads to merge reads into longer contigs (3) Encode reads with regard to the corresponding contigs. Reference-free compression achieved high compression ratio on single FASTQ file. However, these methods focus on the inter-similarity in the same reads set, the exploration of similarity between different reads sets is still lacking. For compression of large collections (databases) of reads sets, optimization for the current one-by-one approach is needed.

2.3 De Novo Assembly and Error Correction of Reads Data

2.3.1 De Novo Assembly

De novo sequence refers to constructing genomes from DNA fragments (short or long reads). The de novo assembly of reads data is of prime importance for the downstream analysis of NGS data (Miller, Koren & Sutton 2010).

For assembly of short reads, de Bruijn graph is introduced (Pevzner, Tang & Waterman 2001) and widely used. In de Bruijn graph, each unique k-mer is represented as a node and adjacent k-mers share overlapping with length $k-1$, this presentation of data displays all k-mers and $k-1$ length overlapping explicitly, in addition, as the structure is based on k-mers instead of reads, the high redundancy in reads data is handled naturally without increasing the number of nodes.

The assembly tools normally generate a de Bruijn graph that represents all k-mers and $k-1$ length overlappings in reads set first, then compute the original genomic sequence via extracting path through the de Bruijn graph (Minoche, Dohm & Himmelbauer 2011).

Velvet (Zerbino & Birney 2008) attached each node in de Bruijn graph with a twin node to represent the reverse complement k-mers, then remove the tips, bubbles and error corrections in graph, which shows great performance on very short reads (25-50bp) data.

IDBA (Peng, Leung, Yiu & Chin 2010) designs an iterative de Bruijn graph that iterates from small to large k values to achieve a balance between handling erroneous reads and repeat regions, which produces longer contigs with similar accuracy.

The strategy of SOAPdenovo (Li, Zhu, Ruan, Qian, Fang, Shi, Li, Li, Shan, Kristiansen et al. 2010) specializes in de novo assembly of large genomes from short reads sequences: error correction is conducted before de Bruijn graph construction, then low-coverage nodes and bubbles are

removed from constructed de Bruijn graph. SOAPdenovo2 (Luo, Liu, Xie, Li, Huang, Yuan, He, Chen, Pan, Liu et al. 2012) is an updated version with improvements in resolving repeat regions to reduce memory consumption in graph construction.

SPAdes (Bankevich, Nurk, Antipov, Gurevich, Dvorkin, Kulikov, Lesin, Nikolenko, Pham, Prjibelski et al. 2012) selects reads passed by h-path in standard de Bruijn graphs with $k \in [a, b]$ to construct standard de Bruijn with $k = b$, which is referred as multi-sized de Bruijn graph. Then it identifies bulge/tip/chimeric by detecting parallel paths in de Bruijn graph and removes the corresponding reads. After that, accurate distance between k-mers are derived via joint analysis of distance histograms, at last, a paired assembly graph (inspired by partial de Bruijn graph) is constructed for final assembly.

De Bruijn graph and its variants are ideal for de novo assembly of short reads data. However, de Bruijn based approach assumes that most k-mers from the genome are preserved in multiple reads, which does not hold for long sequencing reads.

De novo assembly of long reads data is another essential topic for sequence assembly. SGA (Simpson & Durbin 2012) uses the overlap-based string graph model of assembly. Canu (Koren et al. 2017) applies sparse graphical construction and graphical fragment assembly (Li 2016) to improve the performance on single-molecule sequence data. Wengan (Di Genova, Buena-Atienza, Ossowski & Sagot 2019) combines the exploitation of short and long reads data to tackle assembly contiguity as well as consensus quality. Flye (Kolmogorov, Yuan, Lin & Pevzner 2019) generates disjoints that represent concatenations of multiple disjoint genomic segments instead of contigs, then constructs repeat graph and extracts contigs that formed by the path in graph. Peregrine (Chin & Khalak 2019) index reads via sparse hierarchical minimizers to solve the computational burden of long reads assembly. Recently, the advanced technologies produced promising long reads data such as HiFi (Wenger, Peluso, Rowell, Chang, Hall, Concepcion, Ebler,

Fungtammasan, Kolesnikov, Olson et al. 2019), which launched a new de novo assembly era. Garg et al. (Garg, Fungtammasan, Carroll, Chou, Schmitt, Zhou, Mac, Peluso, Hatas, Ghurye et al. 2019) combine accurate HiFi data and long-range Hi-C data for chromosome-scale assembly. HiCanu (Wenger et al. 2019) is a modification of Canu to leverage the full potential of HiFi reads data via homopolymer compression.

2.3.2 Error Correction

Existing tools normally attempt to replace less-frequent nucleotides with more-frequent ones based on sufficient coverage. The error correction methods can be classified into three types: k-spectrum based, suffix tree/array based and multiple sequence alignment(MSA) based.

k-mer frequency spectrum methods such as Reptile (Yang, Dorman & Aluru 2010), Hammer (Medvedev, Scott, Kakaradov & Pevzner 2011), Musket (Liu, Schröder & Schmidt 2013), Blue (Greenfield, Duesing, Papanicolaou & Bauer 2014) slide a fixed-sized k-mer window along each entire reads to generate k-mer spectrum. Then a threshold of number of k-mer appearances is set to determine solid k-mers (also referred as error-free k-mers) and weak k-mers (erroneous k-mers). Finally, these methods remove the weak k-mers and correct reads based on solid k-mers. Bless (Heo, Wu, Chen, Ma & Hwu 2014) detects solid k-mers via hash table and stores duplicate k-mers in Bloom filters to reduce memory usage, which enables usage of longer k-mer. Lighter (Song, Florea & Langmead 2014) samples k-mers randomly, using a pair of Bloom filters to store the k-mers and solid k-mers of each read. RECKONER (Długosz & Deorowicz 2017) applies KMC2 (Deorowicz, Kokot, Grabowski & Debudaj-Grabysz 2015) for k-mer counting and core correction strategy from BLESS2 (Heo, Ramachandran, Hwu, Ma & Chen 2016) with improvement in utilizing quality score. The work of (Zhao, Xie, Bai, Chen, Wang, Zhang, Wang, Zhao & Li 2018) models the frequencies k-mers in Gamma and mixture of Gaussian distribution to determine solid k-mers, then constructs a Bloom filter based on solid k-mers for error correction.

Suffix tree/array-based methods (e.g. Hitec (Ilie, Fazayeli & Ilie 2011) Racer (Ilie & Molnar 2013), Fiona (Schulz, Weese, Holtgrewe, Dimitrova, Niu, Reinert & Richard 2014)) can be considered as a generalization of k-spectrum with flexible k-values. A suffix array/tree of all reads suffixes is built and the errors are corrected by using the k-mer frequency weights associated with the suffix tree nodes. However, the memory requirement of these methods restricts the applicability of these methods.

Although the k-spectrum methods are the easiest and fastest solutions, they are unable to reliably detect the low coverage distinguish errors from polymorphisms (Harismendy, Ng, Strausberg, Wang, Stockwell, Beeson, Schork, Murray, Topol, Levy et al. 2009, Treangen & Salzberg 2012). Other EC tools tend to detect and correct errors based on multiple sequence alignment (MSA) or hybrid methodologies.

In MSA based methods, each read r is performed multiple alignment of reads that selected with different strategies, such as share common or similar k-mers with r. MuffinKmeans (Alic, Tomás, Torres, Medina & Blanquer 2014) groups reads based on spectral clustering (Von Luxburg 2007) before applying MSA. Karect (Allam et al. 2015) employs an improved heuristic method that allows mismatch/indels to select candidate reads for each read r to conduct multiple alignment on r and set a weight for each r, then store the alignment result in partial order graph (POG). FMOE (Huang & Huang 2017) first identifies overlapping reads by aligning a query read simultaneously against multiple reads compressed by FM-index, then corrects sequencing errors by k-mer voting from overlapping reads only. MEC (Zhao, Chen, Li, Jiang, Wong & Li 2017) applies a two-layered MapReduce technique to map reads with the identical k-mer into the same group, then performs MSA on each group of reads.

Both error correction tools and the built-in procedure of de novo assembly tools detect and correct the errors in data. Error correction tools tend to utilize the frequency of k-mers in reads data, while de novo assembly tools detect the erroneous structures in de Bruijn graph and modify the

corresponding reads.

2.4 Summary

This chapter has reviewed existing methods for addressing the following problems: reference-based compression of genome data, reference-free compression of short reads data, de Novo assembly and error correction of reads data. In addition, some limitations of existing method are discussed.

Chapter 3

Sketch Distance-based Clustering of Chromosomes for Large Genome Database Compression

3.1 Introduction

In this chapter, we focus on unsupervised learning for the compression of multiple genomes.

We propose ECC (Tang et al. 2019), an **E**fficient **C**lustering-based reference selection algorithm for the **C**ompression of genome databases. Instead of using a fixed reference sequence by the literature methods, our idea is to cluster the genome sequences of the database into subsets such that genomes within one subset are more similar than the genomes in the other subsets, and then select the centroid genome as reference within each cluster for the compression. Then we select a final reference to compress remaining centroid sequences.

We use the MinHash technique (Broder 1997, Ondov, Treangen, Melsted, Mallonee, Bergman, Koren & Phillippy 2016) to measure the distance

between sequences to construct a distance matrix of the genomes for the clustering. For a genomic sequence L (e.g., a chromosome sequence), MinHash first generates a set of constituent k-mers of L. Then the k-mers are mapped to distinct hash values through a hash function H (the set of hash values is denoted by $H(L)$). Then a small q number of the minimal hash values is sorted. This set of q smallest hash values is called a *sketch* of $H(L)$ (Ondov et al. 2016), denoted by $Sk(H(L))$. So, MinHash can map a long sequence (or a sequence set) to a reduced representation of k-mers which is called a sketch. Given two long sequences L_1 and L_2, MinHash uses some set operations on the sketches of L_1 and L_2 to efficiently estimate the distance between the original L_1 and L_2 under some error bounds. Recent studies have shown that sketch distance and MinHash are very effective in clustering similar genomic sequences with wide applications to genome assembly (Berlin, Koren, Chin, Drake, Landolin & Phillippy 2015), metagenomics clustering (Rasheed & Rangwala 2013), and species identification of whole genome sequences (Ondov et al. 2016).

The main steps of our ECC method are as follows:

1. Construct a distance matrix of the n genome sequences using the pairwise sketch distance method Mash (Ondov et al. 2016).

2. Utilize unsupervised learning to cluster the genomes based on the distance matrix, determine one reference sequence within each cluster and take the remaining ones as target sequences.

3. Compress the target sequences within each cluster by a reference-based compression algorithm, and select a final reference sequence for the compression of the remaining reference sequences.

The key differences between ECC and other compression schemes for sequence databases such as MSC (Cheng et al. 2015) and RCC (Cheng et al. 2017) include: (i) our estimation on pairwise sequence distances is based on the sketch distance of the reduced k-mer sets (Broder 1997) instead of the Euclidean distance between vectors of k-mer frequencies (Cheng et al. 2017);

(ii) our initial setting of the centroid in the clustering is not randomly as by RCC, but determined by the analysis on the whole database; (iii) the reference selection within the clusters is also decided by the clustering method instead of the reconstruction of the original target genome set by RCC.

The first difference implies that our approach is faster than the other methods and makes the clustering applicable to large sequence sets (RCC or MSC is limited to only short genome sequences due to its extremely high computational complexity). The second point of difference prevents the convergence to a local minimum for the **K**-medoids clustering method and makes the clustering results stable. The third point implies that our method compresses sequence set without the need to record additional information in the result. GDC2 is so far the best reference-based algorithm for the compression of the Human 1000 Genomes Database. The reference was selected external to the database. However, when the user is unfamiliar with the similarity between sequences in given set, the selection of one fixed reference sequence may result in very poor performance on dissimilar target sequences and a long running time in the compression. While the reference selection by ECC is decided by the clustering step, and all the references are internal genomes of the database which are required to be compressed.

In the experiments, we compared the performance on genome databases between the straightforward reference-fixed compression approach and our clustering approach ECC for the state-of-the-art reference-based compression algorithms. Our approach achieved 22.05% compression gain against the best case of the reference-fixed compression approach on a set of 60 human genomes collected from NCBI, where the compression ratio increases from 351.74 folds to 443.51 folds. On the union set of the Human 1000 Genomes Project and the 60-genome NCBI dataset, the compression ratio increases from 2919.58 folds to 3033.84 folds. Similar performance improvement over the rice genome database has also been observed.

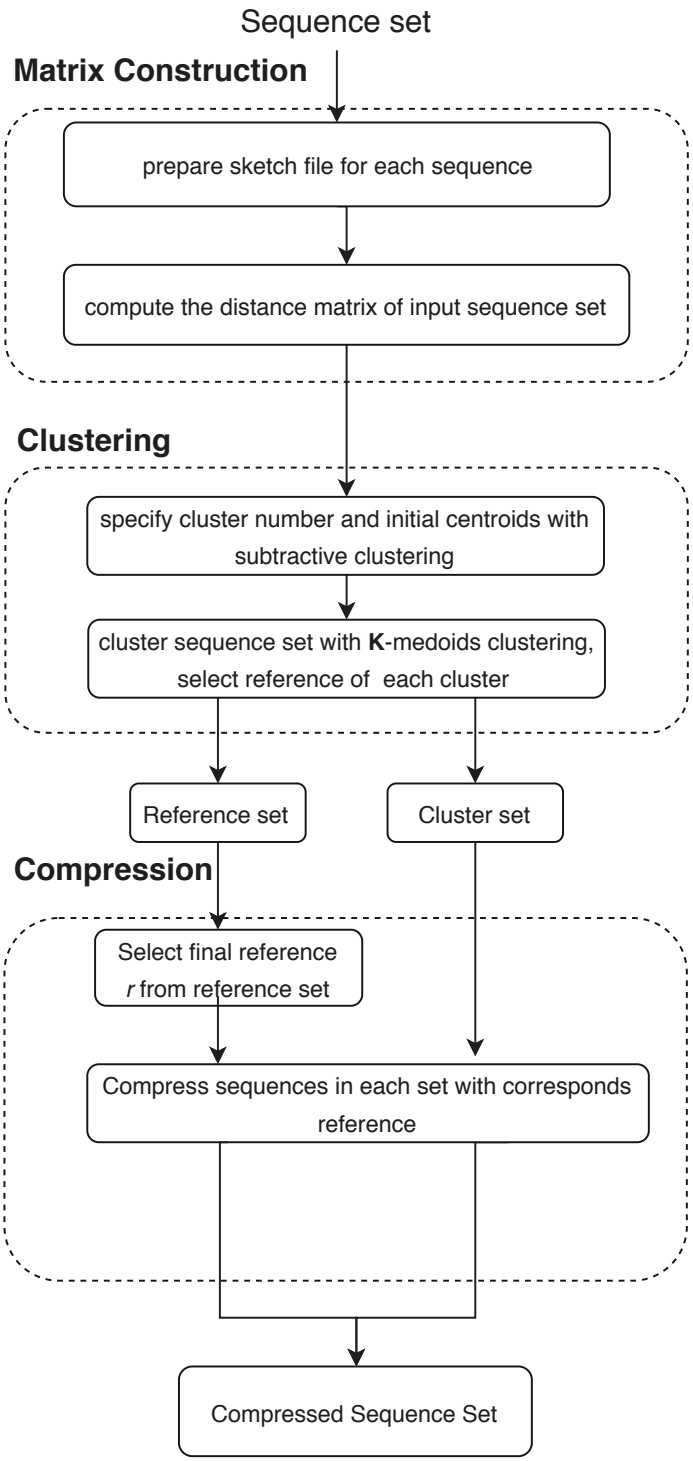


Figure 3.1: Workflow of our ECC approach for compressing a database of genomic sequences

3.2 Method

Our algorithm ECC consists of three stages: distance matrix construction for chromosome sequences, chromosome sequences clustering and chromosome sequences compression. A schematic diagram of the method is shown in Figure 3.1.

3.2.1 Construction of Distance Matrix for a Set of Chromosome Sequences

Let $\mathcal{S} = \{S_1, S_2, \cdots, S_n\}$ be a collection of genomic sequences (i.e., a genome database or a chromosome database). We use a MinHash toolkit called Mash (Ondov et al. 2016) to compute pairwise sketch distances of the sequences to form a distance matrix. By the tool Mash, a sequence S_i is firstly transformed into the set of its constituent k-mers, then all the k-mers are mapped to distinct 32-bit or 64-bit hash values by a hash function. Denote the hash values set of the constituent k-mers set from S_i as $H(S_i)$, and denote the set of q minimal hash values as $Sk(H(S_i), q)$, which is a size-reduced representative of $H(S_i)$, and is called a sketch of $H(S_i)$. For two hash-value sets A and B, the Jaccard index of A and B is defined as $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$, and it can be estimated by $J'(A, B) = \frac{|Sk(A \cup B, q) \cap Sk(A, q) \cap Sk(B, q)|}{|Sk(A \cup B, q)|}$. The sketch distance d_{sk} between two sequences S_i and S_j is defined as

$$d_{sk}(S_i, S_j) = -\frac{1}{k} \ln \frac{2 * J'(H(S_i), H(S_j))}{1 + J'(H(S_i), H(S_j))} \tag{3.1}$$

where the Jaccard index between S_i and S_j is approximately computed using the sketches of $H(S_i)$ and $H(S_j)$. We construct a distance matrix M for sequence set $\mathcal{S}$ with size n. M is a square matrix with dimension $n \times n$ that contains all the pairwise sketch distances between these genomic sequences. The elements of M are defined as:

$$M_{ij} = \begin{cases} 0 & i = j \\ d_{sk}(S_i, S_j) & i \neq j \end{cases} \qquad i, j \in [1, n] \tag{3.2}$$

It is clear that M is a symmetric matrix (i.e., $M_{ij} = M_{ji}$). It can also be understood that the calculation of the sketch distance between two long sequences is much more efficient than the calculation by using k-mer feature vector direct comparison. The efficiency becomes significant, especially in the construction of the whole distance matrix M.

3.2.2 Clustering of Chromosomes from the Distance Matrix

Clustering is the process of grouping a set of samples into a number of subgroups such that similar samples are placed in the same subgroup. Here our clustering is to ensure a higher similarity between each reference-target pair for achieving an outstanding compression performance. An important step in the process of clustering is to determine the number of clusters in the data. We take a subtractive clustering approach (Chiu 1994, Dhanachandra, Manglem & Chanu 2015) to decide the number of clusters in the distance matrix M, and then use the **K**-medoids clustering method (Park & Jun 2009) to group the n number of genomic sequences into **K** number of clusters.

Subtractive Clustering to Determine the Number of Clusters K

Most clustering algorithms require the number of clusters as a parameter. However, the cluster number for a set of genomic sequences is normally unknown. We utilize a modified subtractive clustering algorithm to specify the cluster number.

Subtractive clustering is an extension of the Mountain method (Yager & Filev 1994). It estimates cluster centroid based on the density of points in the data space. We apply the exponential function for the Mountain Value Calculation. Given a sequence set $\mathcal{S}$, the corresponding sketch distance matrix M with dimension $n \times n$ and a threshold percentage $\epsilon \in (0, 1)$, the process to determine the number of clusters is:

1. Create the empty cluster centroid set $\mathcal{O}$. Compute the mountain value

of each sample S_i:
$Mt(S_i) = \sum_{j=1}^{n} e^{-M_{ij}}$

2. Let $o = \text{argmax}_{i=1}^{n} Mt(S_i)$, add S_o to $\mathcal{O}$.

3. Update the mountain value of each remaining sequence by:
$Mt(S_i) = Mt(S_i) - e^{-M_{io}}$

4. Repeat step 2 and step 3 until $Mt(S_i) < \epsilon Mt_{max}$ or $|\mathcal{O}| \geq \sqrt{n}$.

5. Return centroids set $\mathcal{O}$ and cluster number $\mathbf{K}$= $|\mathcal{O}|$.

K-medoids Clustering of the Collection of n Genomic Sequences

K-medoids is a partition-based cluster analysis method. **K**-medoids iteratively finds the **K** centroids and assigns every sample to its nearest centroid (Park & Jun 2009), which is similar to **K**-means (MacQueen et al. 1967) but more effective for handling outliers. It divides the data set $\mathcal{S}$ into **K** non-overlapping subgroups $\mathcal{C}$ that contains every element of $\mathcal{S}$ and selects a centroid sequence O_i from each subgroup:

Definition 3.1 *For a set of sequence* $\mathcal{S} = \{S_1, \cdots, S_n\}$*, the corresponding cluster set* $\mathcal{C} = \{C_1, C_2, \cdots, C_K\}$ *and centroid sequence set* $\mathcal{O} = \{O_1, O_2, \cdots, O_K\}$ *satisfies the following requirements:* $C_i \subseteq \mathcal{S}$,$C_1 \cup C_2 \cup \cdots \cup C_K = \mathcal{S}$,$C_i \cap C_j = \emptyset$ *for* $i \neq j$, $O_i \in C_i$.

The cluster set $\mathcal{C}$ is determined via minimizing the cost function λ as follows:

$$\lambda(\mathcal{S}) = \sum_{i=1}^{K} \sum_{S_a \in C_i} d_{sk}(S_a, O_i)$$

Though **K**-medoids is efficient, it has some drawbacks. The clustering result highly depends on the setting of the initial centroids. To improve the stability and quality of clustering result, instead of arbitrarily selecting the initial centroids by the standard **K**-medoids, we use the centroid set $\mathcal{O}$ as computed by subtractive clustering in the previous section.

Given a sequence set $\mathcal{S}$, sketch distance matrix M, cluster number $\mathbf{K}$ and centroid sequence set $\mathcal{O}$, the $\mathbf{K}$-medoids proceeds by the following steps:

1. Set $\mathcal{O}$ as the initial centroid sequence set.
2. Associate each S_i to the centroid O_j with minimum sketch distance, also associate S_i to cluster C_j.
3. Recalculate the new centroid of each cluster based on its elements:

$$O_j = \underset{S_a \in C_j}{\operatorname{argmin}} \sum_{S_b \in C_j} d_{sk}(S_a, S_b)$$

4. Repeat steps 2 and 3 until $\mathcal{C}$ and $\mathcal{O}$ no longer change or reach a pre-set number of iterations.
5. Return cluster set $\mathcal{C}$ and cluster centroid set $\mathcal{O}$.

3.2.3 Compression

Chromosome sequences set $\mathcal{S}$ is compressed based on the cluster set $\mathcal{C}$ and centroids set $\mathcal{O}$ computed by $\mathbf{K}$-medoids. First, use O_i as the reference sequence for the other sequences in cluster C_i. Then select a final reference R from the centroid set as the reference for the other centroid sequences:

$$r = \underset{O_i \in \mathcal{O}}{\operatorname{argmin}} \sum_{O_j \in \mathcal{O}} d_{sk}(O_i, O_j)$$

In detail, all the sequences in cluster C_i is compressed using O_i as the reference sequence except O_i itself. Then all the reference sequences except R are compressed using R as the reference sequence. The final reference R can be compressed by the block-sorting compression (bsc) algorithm (http://libbsc.com/) or other reference-free compression algorithms.

All non-centroids sequences will be compressed with centroid sequences as reference and centroid sequences (except R) will be compressed with R as reference, only one final reference sequence R will remain uncompressed.

It is clear that the same number of sequences is compressed in ECC as in straightforward approach.

All reference-based compression algorithms can take this clustering approach to compress a set of genomic sequences. The pseudo-code of our compression method is presented in Algorithm 1.

Algorithm 3.1 Compression of n genomic sequences

Input Sequence set $\mathcal{S} = \{S_i\}_{i=1}^{n}$. Distance Matrix M. Cluster set $\mathcal{C} = \{C_i\}_{i=1}^{K}$, Cluster centroid set $\mathcal{O} = \{O_i\}_{i=1}^{K}$.

```
1: L ←an array with K integers
2: count ← 1
3: for i = 1 to n do
4:     if S_i ∉ O then
5:         g ← the id of cluster that corresponds to S_i
6:         compress S_i with O_g
7:     else
8:         L[count] ← i
9:         count ← count + 1
10: r ← argmin_{i∈L} Σ_{j∈L} M_ij
11: R ← S_r
12: for i = 1 to K do
13:     if L[i] ≠ r then
14:         compress S_{L[i]} with R
```

Decompression

The decompression process is the reverse process of compression. All the sequences except R require a reference to decompress. Firstly, R is decompressed; then the reference sequence of each cluster is decompressed by R, all the remaining sequences in the cluster are decompressed by the reference sequence in its cluster. As the process is invertible, the compression

scheme is lossless as long as the used reference-based compression algorithm is lossless.

3.2.4 Data

To assess the performance of our proposed method ECC, we compare the compression ratio based on ECC result with the reference-fixed compression approach on multiple genome databases.

These include: a set of 60 human genome sequences (denoted by dataset-60) from National Center for Biotechnology Information (NCBI) [1]with a file size of 171GB, a set of 1152 human genome sequences (dataset-1152) from the 1000 Genomes Project (Siva 2008) and NCBI with a file size of 3,128GB, and a set of 2818 rice genomes (dataset-2818) from the 3000-rice project (Wang et al. 2018) with a file size of 1,012GB.

3.3 Experiment and Results

This section describes our experimental results on dataset-60, dataset-1152 and dataset-2818 to evaluate the performance of our approach. In particular, the compression ratio and running time of our algorithm are presented and discussed in comparison with the reference-fixed compression approach.

3.3.1 Test Methodology

Our algorithm was implemented in the C++11 language. All experiments were conducted on a machine running Red Hat Enterprise Linux 6.7 (64 bit) with 2 × Intel Xeon E5-2695 processors (2.3GHz,14 Cores), 128 GB of RAM.

Six state-of-the-art reference-based compression algorithms were tested on the three genome databases to understand the performance improvement achieved by our clustering approach in comparison with the reference-fixed compression approach. These compression algorithms are HiRGC (Liu,

[1]https://www.ncbi.nlm.nih.gov/genome

Table 3.1: Compression ratio for the H. sapiens dataset-60 (171GB)

Reference	Compression ratio with algorithm					
	HiRGC	iDoComp	GDC2	ERGC	NRGC	SCCG
GCA_000004845	339.80	184.20	238.98	11.00	122.67	225.35
hg19	346.80	26.78	242.60	128.11	137.41	265.03
YH	351.74	134.13	237.39	108.26	123.24	228.20
GCA_000252825	241.01	92.65	230.45	102.62	176.08	122.25
Huref	245.79	140.84	224.47	69.59	177.85	123.26
ECC clustering result	**443.51**	**238.68**	**248.11**	**293.24**	**184.13**	**313.60**
Ratio gain*	22.05%	22.83%	2.22%	56.31%	3.41%	15.49%

Bold text indicates the highest compression ratio of an algorithm, blue text indicates the **best** case of fixed single reference compression result.
*The ratio gain of ECC against the **best** case.

Table 3.2: Compression ratio gain of ECC against different cases on H. sapiens dataset-60 (171GB)

Reference	Compression ratio with algorithm					
	HiRGC	iDoComp	GDC2	ERGC	NRGC	SCCG
GCA_000004845	24.70%	22.83%	3.68%	96.25%	33.38%	28.14%
hg19	23.14%	88.78%	2.22%	56.31%	25.37%	s15.49%
YH	22.05%	43.81%	4.32%	63.08%	33.07%	27.23%
GCA_000252825	46.59%	61.18%	7.12%	65.00%	4.37%	61.02%
Huref	45.53%	41.00%	9.53%	76.27%	3.41%	60.70%

Peng, Wong & Li 2017), iDoComp (Ochoa et al. 2014), GDC2 (Deorowicz, Danek & Niemiec 2015), ERGC (Saha & Rajasekaran 2015),NRGC (Saha & Rajasekaran 2016) and SCCG (Shi, Chen, Luo & Chen 2018). All the algorithms that are compatible with multi-cores computing were executed with 4 cores.

We also attempted to test the performance of RCC (Cheng et al. 2017) on the same genome databases. However, it was not runnable for the compression of long genome sequences (such as human and rice) due to its time complexity— RCC was taking longer than 10 hours to compress only four human genome sequences.

For GDC2, as its two-level compression structure tends to compress all the target sequences using the same reference, we compress the datasets using the final reference selected by ECC, and the compression order of GDC2 is also adjusted in accordance with the ECC clustering result.

As mentioned before, the performance of a reference-based algorithm

on the NGS dataset is highly dependable on the option of the reference sequence. To reduce the variance from an arbitrary selection, we randomly selected multiple reference sequences from the target dataset and obtain the compression performance with each of them for the compression algorithms (the randomly selected reference file itself is not compressed, so all experiments compress the same number of genome sequences).

To measure the performance improvement, we denote the compression ratio with fixed single reference as C_S and the compression ratio on same dataset with ECC as C_E, and introduce a relative compression ratio gain as:

$$G = (1 - \frac{C_S}{C_E}) \times 100\%$$

A larger value of compression ratio gain indicates a more significant improvement.

3.3.2 Gains of Compression Performance

Our proposed ECC method outperforms the reference-fixed compression approach in all cases on dataset-60 (see Table 3.1). The compression gains against the best results by the reference-fixed compression approach are 22.05%, 22.83%, 2.22%, 56.31%, 3.41%, 15.49% for HiRGC, iDoComp, GDC2, ERGC, NRGC, and SCCG respectively. On dataset-60, HiRGC, iDoComp, ERGC and SCCG gained more compression improvement, while the effect of ECC on NRGC and GDC2 is relatively smaller. Moreover, HiRGC, iDoComp, SCCG and GDC2 achieved higher compression ratio on this database than ERGC and NRGC in general.

We added the 1092 human genomes from the 1000 Genome Project to dataset-60 (denoted by H. sapiens dataset-1152) and conducted another round of experiments. Performance details are summarized in Table 3.3 for HiRGC, iDoComp and GDC2 which are the three algorithms of the highest compression performance on dataset-60. The overall compression performance is higher than on dataset-60. Through ECC, iDoComp gained 15.86% compression performance against the best reference-fixed

compression case, while HiRGC gained 7.95%. The ratio gain of GDC2 is only 3.77%, but more importantly, ECC helped GDC2 avoid 3 of the 7 time-consuming cases in the reference-fixed approach.

Table 3.3: Compression ratio on H. sapiens dataset-1152 (3128GB)

Reference	Compression ratio with algorithm		
	HiRGC	iDoComp	GDC2
HG00096	991.77	485.35	2919.58
NA18856	889.32	437.05	2805.19
GCA_000004845	784.84	53.94	2901.44
GCA_000252825	504.41	114.40	2897.76
GCA_000365445	13.07	/	/
hg19	1046.84	68.36	/
hg38	826.31	52.03	/
result of ECC	**1137.21**	**576.84**	**3033.84**
Ratio gain*	7.95%	15.86%	3.77%

'/' indicates a running time longer than 500 hours. Bold text indicates the highest compression ratio of an algorithm, blue text indicates the **best** case of fixed single reference compression result.
*The ratio gain of ECC against the **best** case.

On the rice genome dataset-2818, through our ECC clustering approach, HiRGC gained 13.89% compression performance against the best case by the reference-fixed compression approach, iDoComp gained 21.22%, and GDC2 gained 2.48% (Table 3.5). The compression ratio gain of HiRGC is more stable than on the first two human genome databases. A reason is that all the genomes in the rice database were aligned to the sequenced rice cultivars:

Table 3.4: Compression ratio gain of ECC against different cases on H. sapiens dataset-1152 (3128GB)

Reference	Compression ratio with algorithm		
	HiRGC	iDoComp	GDC2
HG00096	12.79%	15.86%	3.77%
NA18856	21.80%	24.23%	7.54%
GCA_000004845	30.99%	90.65%	4.36%
GCA_000252825	55.64%	80.17%	4.49%
GCA_000365445	98.85%	/	/
hg19	7.95%	88.15%	/
hg38	27.34%	90.98%	/

93-11 (indica variety) (Stein, Yu, Copetti, Zwickl, Zhang, Zhang, Chougule, Gao, Iwata, Goicoechea et al. 2018). Hence this dataset has a higher inter-similarity and the variance from the random selection of the fixed reference is smaller. Compression ratio gain of ECC against different cases on H. sapiens dataset-1152 (3128GB)

From these comparisons, we can understand that our ECC clustering approach can make significant compression improvement for most of the state-of-the-art algorithms and can avoid selecting some inappropriate references such as the 3 extremely time-consuming cases of GDC2 on the human dataset-1152.

3.3.3 Speed Performance

Running time is an essential factor for measuring the applicability of an algorithm in the compression of large-scale genome databases.The running

Table 3.5: Compression ratio on the Oryza sativa dataset-2818(1,012GB)

Reference	Compression ratio with algorithm		
	HiRGC	iDoComp	GDC2
B035	79.31	77.62	529.40
CX319	73.76	81.55	537.09
IRIS_313-10010	69.93	64.74	519.43
IRIS_313-10776	70.97	77.10	533.81
IRIS_313-9937	71.39	66.42	535.31
result of ECC	**92.10**	**103.52**	**550.77**
Ratio gain*	13.89%	21.22%	2.48%

Bold text indicates the highest compression ratio of an algorithm, blue text indicates the **best** case of fixed single reference compression result.
*The ratio gain of ECC against the **best** case.

Table 3.6: Compression ratio gain of ECC against different cases on Oryza sativa dataset-2818(1,012GB)

Reference	Compression ratio with algorithm		
	HiRGC	iDoComp	GDC2
B035	13.89%	25.02%	3.88%
CX319	19.91%	21.22%	2.48%
IRIS_313-10010	24.07%	37.46%	5.69%
IRIS_313-10776	22.94%	25.52%	3.08%
IRIS_313-9937	22.49%	35.84%	2.81%

Table 3.7: Reference selection time of ECC (in hours)

Dataset	dataset-60	dataset-1152	dataset-2818
number of genomes	60	1152	2818
total running time	0.023	0.830	0.759

Table 3.8: Compression time of each algorithm on the three datasets

Algorithm	Compression time (in hours) for					
	dataset-60		dataset-1152		datset-2818	
	reference-fixed	ECC	reference-fixed	ECC	reference-fixed	ECC
HiRGC	1.18	0.98	15.12	13.94	2.91	2.82
iDoComp	6.54	2.82	102.94	29.77	15.58	10.34
GDC2	110.73	117.82	129.24	126.43	25.29	23.61

The time by the reference-fixed approach is the average running time of several fixed single-reference cases by each algorithm, please see the supplementary file for the time range of all the cases and compression time by ERGC, SCCG and NRGC.

time of ECC includes two parts: reference selection time (only depending on the input sequence set) and the compression time (depending on the input sequence set and the reference-based compression algorithm).

As shown in Table 3.7, ECC took 0.02, 0.83, 0.76 hours on the reference selection part for dataset-60, dataset-1152 and rice genome dataset-2818 respectively. But the compression time for these three datasets are 0.98, 13.94, 2.82 hours (Table 3.8) by HiRGC, which is the fastest algorithm in the compression. The reference selection time is much shorter than the sequence compression time.

We have also observed that the total time of reference selection and compression by ECC is highly competitive with the reference-fixed compression approach. In fact, the compression time via ECC after the reference selection is shorter than the compression time of the reference-fixed compression in most cases except GDC2 on the dataset-1152 (Table 3.8).

3.4 Summary

In this chapter, we introduced ECC, a clustering-based reference selection method for the compression of genome databases. The key idea of this method is the calculation of a MinHash sketch distance between chromosome sequences to group the chromosome sequences into subsets of similar sequences. Within each cluster, the reference chromosome is best updated according to the shortest sketch distance to the centroid chromosome. This algorithm is universal for genome sequence sets of the same species. We have demonstrated that the six state-of-the-art reference-based compression algorithms all achieved a substantial improvement after the clustering of the genome sequences, with similar amounts of compression time consumed by the reference-fixed approach.

Availability of data and materials

All data associated with this study are available in the Supplementary Data file. The C++ codes of our algorithm are available at https://github.com/Tao-Tang/ECC.

Chapter 4

Unsupervised Learning for Compression of Short Reads Databases

In the section, we focus on unsupervised learning for the compression of large-scale reads datasets.

4.1 Introduction

We propose a novel clustering method (named MRC — **M**ultiple **R**eads Sets **C**lustering) for compressing reads datasets databases. MRC constructs feature vectors based on the minimizer frequencies of each reads set, then applies K-means clustering recursively to divide the reads datasets into subgroups with higher inter-similarity than the original reads datasets. Then the compression of these clustered reads datasets is carried out group-by-group instead of one-by-one separately. Compared to the previous straightforward one-by-one compression for large collections of FASTA files, our method utilizes the similarity between reads in different reads to achieve better performance.

We evaluate MRC on various benchmark databases to demonstrate that

the method improves the compression ratio as compared to the original algorithms in all cases.

4.2 Theoretical Analysis

For a set of n number of reads data sets to be compressed. An easy question is whether we can concatenate the n data sets together as a single large data set for compression. A difficulty is the extremely high usage of main memory, with no guarantee of compression performance improvement. A challenging question is how to convert a large reads data set into a characteristic feature vector and how to cluster these n vectors to find small groups of similar data sets to merge for compression.

Intuitively in theory, merging similar data sets for compression is potentially useful for performance improvement. Denote a reads set as Y, the assembled consensus sequence(s) from Y as $\overline{Y}$. The key idea of reference-free compression is to merge these reads into consensus sequence(s) and record each read with regard to the consensus sequence(s). Denote the information of reads set Y after compression as $I(Y), I(Y) = I(\overline{Y}) + I(Y|\overline{Y})$. When Y is split into two sets Y_1 and Y_2. Then $I(Y_1) + I(Y_2) = I(\overline{Y_1}) + I(Y_1|\overline{Y_1}) + I(\overline{Y_2}) + I(Y_2|\overline{Y_2})$. Under the ideal condition (optimal solution for consensus sequences construction and no correction for mismatching bases), the consensus sequence(s) $\overline{Y_1}, \overline{Y_2} \subseteq \overline{Y}, I(\overline{Y}) = I(\overline{Y_1}) + I(\overline{Y_2}) - I(\overline{Y_1} \cap \overline{Y_2}) \leq I(\overline{Y_1}) + I(\overline{Y_2})$. $I(Y|\overline{Y})$ can be considered as the information of $|Y|$ records, where each record contains the position of one read in $\overline{Y}$ (assume no mismatching and all reads have the same length), with an encoding method such as Lempel-Ziv parsing (Ziv & Lempel 1977), the average record length of each compression can be considered as constant, hence $I(Y|\overline{Y}) \approx I(Y_1|\overline{Y_1}) + I(Y_2|\overline{Y_2})$, $I(Y) \approx I(Y_1) + I(Y_2) - I(\overline{Y_1} \cap \overline{Y_2}) \leq I(Y_1) + I(Y_2)$. Compressing two sets together will reduce the compressed size in comparison with compressing them one-by-one separately, and the amount of reduced size should have a positive relationship with the size of the intersection set of the

consensus sequences generated from the two sets.

Multiple methods have been proposed to search the matched substring between sequence data (Liu, Wong & Li 2020). For reference-free compression algorithms, the focus is on utilizing the similarity consensus and sequence redundancy information among the reads. The main idea is to search the overlapping sub-strings in the pairs of reads through a 'seed-extension' strategy, in which a short common subsequence between two reads are used as the 'seed', then the 'seed' is extended, although there are some other ideas of taking suffix-prefix overlapping (e.g. PgRC), or some without the explicit extending process (e.g. minicom). Another effective search strategy is based on the concept of minimizers (Roberts et al. 2004). A minimizer of a string is the lexicographically smallest k-mer of the string, which is widely used in processing of reads data (Liu, Zhang, Zou & Zeng 2020). It is a useful concept to identify sequence similarity — it has been previously proved (Roberts et al. 2004) that two reads are likely to have a large overlap if they share a k-minimizer.

Hence for a high-performance compression of multiple reads datasets, we should increase the occurrence of k-minimizer redundancy and should combine only those reads datasets of more similar reads together.

With these foundations, we propose to detect the k-minimizer substring for every read in each data set, and use these k-minimizers as features and their frequencies in the data set as feature values to transform each data set into a feature vector. Unsupervised clustering algorithms are then applied to these vectors to find similar data sets and merge them. Table 4.1 is an example to illustrate the high similarity between two sequences (reads) when they share the same minimizer.

A real example (Table 4.2) of the feature vector is presented below (only the first 10 elements of the feature vector displayed). The vector is transformed from a huge reads set named ERR532390, which contains 10831122 reads of length 100 with a total file size of 4.27GB. Each feature value in the vector is the percentage of reads in this data set containing

Two sequences	G	T	A	C	T	G	A	T	T	G	C	A	C	T	G	A
3-mers and 3-minimizers (in bold)	G	T	A						T	G	C					
		T	A	C						G	C	A				
			A	**C**	**T**						C	A	C			
				C	T	G						**A**	**C**	**T**		
					T	G	A						C	T	G	
						A	G	T						T	G	A

Table 4.1: Two sequences that share the same minimizer

Table 4.2: First 10 elements of the feature vector transformed from a reads set

IDs of 8-minimizers	0	1	2	3	4	5	6	7	8	9
Normalized frequency	0.56	0.43	0.61	0.79	0.57	0.15	0.28	0.34	0.74	0.42

the same specific minimizer. For instance, feature values 0.56 and 0.43 suggest that there are 0.56% of the reads in ERR532390 containing minimizer AAAAAAAA (feature ID 0) and 0.43% of the reads containing minimizer AAAAAAAC (feature ID 1), where k=8.

If the amount of common k-mers with similar feature values between two data sets is big, then the two data sets should be merged to hold an excessive proportion of reads containing the same minimizer sub-strings to create immense sequence redundancy for boosting compression performance.

4.3 Method

Let $R = \{R_1, R_2...R_n\}$ be a collection (database) of reads datasets, $R_i = \{r_{i1}, r_{i2}..r_{ij}\}$, where r_{ij} denotes a single read sequence, $r_{ij}[l] \in$\{A,C,G,T,N\} for $l = 1, 2, \ldots, L$ where L is the length of the reads.

The proposed algorithm MRC consists of three stages: transformation of a reads set into a feature vector, clustering of the feature vectors, and group-by-group compression.

1. **Transformation of a reads set into a feature vector**: A feature space with n feature vectors is constructed, each of them is transformed from a reads set R_i using the minimizers as features and the minimizers' frequency as feature values.

2. **Clustering of the feature vectors**: A variant of K-means clustering, that is, a K-means clustering with a limit on cluster size is applied to the feature space to divide the original set R into subgroups.

3. **Group-by-group compression**: The reads datasets in each subgroup are compressed together by state-of-the-art compression algorithms with additional information recorded for lossless decompression.

4.3.1 Transformation of a Reads Set into a Feature Vector

For a reads set R_i, the reads are classified into two types: the reads without 'N' and the others. During transformation, only reads without 'N' is considered. The set of reads without 'N' from R_i is denoted as $\underline{R_i} = \{\underline{r_{i1}}, \underline{r_{i2}}..\}, \underline{r_{ij}}[l] \in \{A, C, G, T\}$. Via an encoding function, each k-mer (subsequence s with length k) in $\underline{r_{ij}}$ can be mapped to a $2 \times k$ bit integers, such as $f(s) = \sum_{a=1}^{a=k} \phi(s[a] \cdot 4^{a-1}), \phi(A) = 0, \phi(C) = 1, \phi(G) = 2, \phi(T) = 3$.

In de novo compression, the overlap search between reads normally focuses on finding one suitable overlapping for merging and encoding (to reduce computation time) instead of searching all possible overlaps among reads. To fit the procedure of de novo compression, we extract one representative k-mer from each read.

The lexicographically smallest k-mer (minimizer) of each read is selected as the representative k-mer for the read, and counts the frequency of each unique k-mer in the reads set to estimate the similarity between reads datasets. Here we use a normalized frequency as feature value. For a reads set R_i, the corresponding feature vector $V_i = \{v_{i1}, v_{i2}..v_{i4^k}\}, v_{ij} = \frac{\text{frequency of minimizer with value}j \text{ in } \underline{R_i}}{|\underline{R_i}|}$, the value of minimizer s is computed using

$f(s) = \sum_{a=1}^{a=k} \phi(s[a]) \cdot 4^{k-a}, \phi(A) = 0, \phi(C) = 1, \phi(G) = 2, \phi(T) = 3$. In order to reduce storage usage, only the frequency of minimizers that appears at least one time is recorded in implementation. Table 4.3 is an example of the minimizers and corresponding feature vector of a set of 10 reads with k=2.

Table 4.3: Example of the minimizers and corresponding feature vector of a set of 10 reads with k=2

Sequence	Minimizer(s)	Mapped value $f(s)$
G T C G C C G A	CC	5
A T T G T C G G	AT	3
G A C G A A T C	AA	0
C T G T T G C A	CA	4
A G G T C C G C	AG	2
G T C A A G C A	AA	0
G G T C T C T A	CT	7
C G A G T T C G	AG	2
T G T T G T G C	GC	9
A C C T A T C G	AC	1

Example of minimizers in a reads set with k=2 and the corresponding values $f(s)$.

Minimizer	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC
Frequency	2	1	2	1	1	1	0	1	0	1

Frequency of each minimizer in above reads set.

Feature	AA	CA	GA	TA	AC	CC	GC	TC	AG	CG	GG	TG	AT	CT	GT	TT
Value	0.2	0.1	0.2	0	0.1	0.1	0	0.2	0	0.1	0	0	0	0	0	0

Normalized feature vector of the above reads set.

To reduce processing time and avoid potential problems such as curse

of dimensionality (Bellman 1966) or overfitting (Tetko, Livingstone & Luik 1995), m features are selected from the original 4^k: the variance of each feature $\{v_{1j}, v_{2j}...v_{nj}\}$ is computed, then m features with highest variance are selected. Finally, a set of m-dimensional n vectors is generated, denoted by $\underline{V} = \{\underline{V}_1, \underline{V}_2 ... \underline{V}_n\}$.

4.3.2 Clustering of the Feature Vectors

Clustering is the process of grouping a set of points (feature vectors) into a number of subgroups such that similar points are placed in the same subgroup. Here our clustering is to ensure higher similarity in each subgroup that are to be compressed together to achieve better performance of compression.

A critical part of this stage is the size of a cluster. As mentioned before, the reads datasets in each cluster will be compressed as a single file by state-of-the-art algorithms, however, the complexity of most algorithms is larger than $O(|R_i|)$, which causes sharply increasing compression time with respect to the number of reads. Hence the total size of each cluster should be limited to control compression time. We apply a variant of K-means clustering to ensure the efficiency and practicability for the method.

Variant of K-means Clustering

K-means is a method that divides n points into K clusters in which each point belongs to the cluster with the nearest centroid (Jain 2010). The main difference between our clustering process and the standard K-means is as follows:

1. The K-means clustering is applied on reads datasets recursively with K set as 2 until the size of all clusters is less than or equal 3.

2. When a cluster with size 1 is generated, one object from another cluster will be tried to add to this size 1 cluster, the criterion is shown in Algorithm 4.1.

The clustering process is shown as Algorithm 4.1.

Algorithm 4.1 Optimized K-means clustering

Input: vector set $\underline{V} = \{\underline{V}_1, \underline{V}_2 ... \underline{V}_n\}$, threshold λ, cluster set C;
$C \leftarrow \emptyset$;

procedure CLUSTERING($\underline{V}$)
    if $|\underline{V}| \leq 3$ **then**
        Add $\underline{V}$ to C;
    else
        $c_1, c_2 \leftarrow$ result of K-means on $\underline{V}$ with K=2
        if $|c_1| + |c_2| \geq 4$ AND ($|c_1| == 1$ OR $|c_2| == 1$) **then**
            $a \leftarrow$ id of cluster with size 1, $b \leftarrow$ id of the other cluster;
            $c_b[min] \leftarrow$ the vector in c_b with minimum distance to
        $c_a[0]$;
            if distance($c_a[0], c_b[min]$) $\leq \lambda \cdot$ minimum distance
        between $c_b[min]$ and other vectors in c_b **then**
                add $c_b[min]$ to c_a;
                remove $c_b[min]$ from c_b;
            Clustering(c_1);
            Clustering(c_2);
Clustering($\underline{V}$);
return C;

4.3.3 Group-by-group Compression

For each cluster $C_i \in C$, C_{ij} represents the jth vector in C_i, which corresponds to one specific reads set in R.

The corresponding reads datasets of each C_i is merged together in order, denote the array of reads number of reads datasets in each cluster by $L = L[1...l]$. The $\sum_{a=1}^{i-1} L[a]$th to $\sum_{a=1}^{i-1} L[a+1]$th reads in the merged set are

Table 4.4: Example of the first 10 features of feature vectors from two clusters in the database of 21 reads datasets

Cluster	Reads set	Feature vector
A	1	0.2990.1040.0070.0350.0490.0320.0090.0250.0230.108
	2	0.3000.1050.0070.0350.0510.0310.0090.0250.0240.109
	3	0.2970.1050.0070.0340.0480.0310.0070.0250.0230.109
B	1	0.1480.0670.0210.0630.0700.0510.0210.0460.0440.102
	2	0.1400.0630.0220.0720.0690.0510.0210.0470.0450.101

the reads of the ith reads set in cluster. The merged set is then compressed under the order preserving mode. The name and read number of each reads set are recorded. The decompression process is the reverse process of the compression. First, the compressed file of each merged set is decompressed, then each reads set is restored using the recorded read number.

4.4 Result and Performance Analysis

Compression experiments were conducted at a computing cluster running Red Hat Enterprise Linux 6.7 (64 bit) with 2 Intel Xeon E5-2695 processors (2.3GHz,14 Cores), 128 GB of RAM. Four state-of-the-art compression algorithms, FaStore (Roguski et al. 2018), SPRING (Chandak et al. 2019), minicom (Liu et al. 2019) and PgRC1.2 (Kowalski & Grabowski 2020*a*), were tested on four collections of reads datasets to understand the performance improvement achieved by our clustering approach in comparison with compressing the set separately via the same compression algorithm. PgRC1.2 and minicom focus on compression of the sequencing reads only, while FaStore and SPRING retain all data in the FASTQ files. All algorithms

Table 4.5: Four collections of reads data sets

Collections	number of reads datasets	total size (in GB)	length of reads
database-17	17	48.57	101
database-21	21	135.68	100
database-50	50	71.04	50
database-100	100	197.97	36

More information of each file can be found in Supplementary Data

were executed with the order preserving mode, other parameters follow the default setting (e.g. 24 threads for minicom, 8 threads for PgRC1.2).

Details about the collections of reads data sets are shown in Table 4.5. The original data were downloaded from http://ftp.sra.ebi.ac.uk/vol1/fastq/. The URLs of each file are listed in Supplementary Data.

4.4.1 Compression Performance Gains

To measure the performance improvement brought by MRC, we define performance *gain* as:

$$gain = \left(1 - \frac{\text{Compressed size with MRC}}{\text{Compressed size of one-by-one compression}}\right) \times 100\%$$

A higher value of *gain* indicates more compression improvement.

Table 4.6 shows the compression results by minicom and PgRC1.2 using the 'one-by-one' approach or the 'MRC' approach. The best compression algorithm PgRC1.2 can compress the 48.57 gigabytes of database-17 into 1.01 gigabytes, achieving 48.09 folds of compression ratio, when the straightforward one-by-one approach is taken. The compression ratio is improved to 54.86

Table 4.6: Compressed file size (in byte) comparison between the straightforward *one-by-one* approach and our MRC approach (with $k = 7$ for the setting of k-minimizer)

Collection	minicom			PgRC1.2		
	one-by-one	with MRC	*gain*	one-by-one	with MRC	*gain*
database-17	1184071680	1053224960	11.05%	1087246442	950620160	12.57%
database-21	6074501120	5800316800	4.47%	6263079421	5722808320	8.63%
database-50	3714529280	3548344320	4.41%	3006844741	2880102400	4.22%
database-100	9593702400	9485285120	1.13%	8828937571	8355092480	5.37%

Collection	SPRING			FaStore		
	straightforward	with MRC	*gain*	straightforward	with MRC	*gain*
database-17	8628849779	8484075520	1.68%	9666709566	9437614080	2.37%
database-21	23029063680	22549555200	2.08%	25241707218	24566077440	2.68%
database-50	13646202880	13519677440	0.93%	15492111069	15372031320	0.78%
database-100	28880547840	28573671680	1.06%	23964221072	23478440608	2.03%

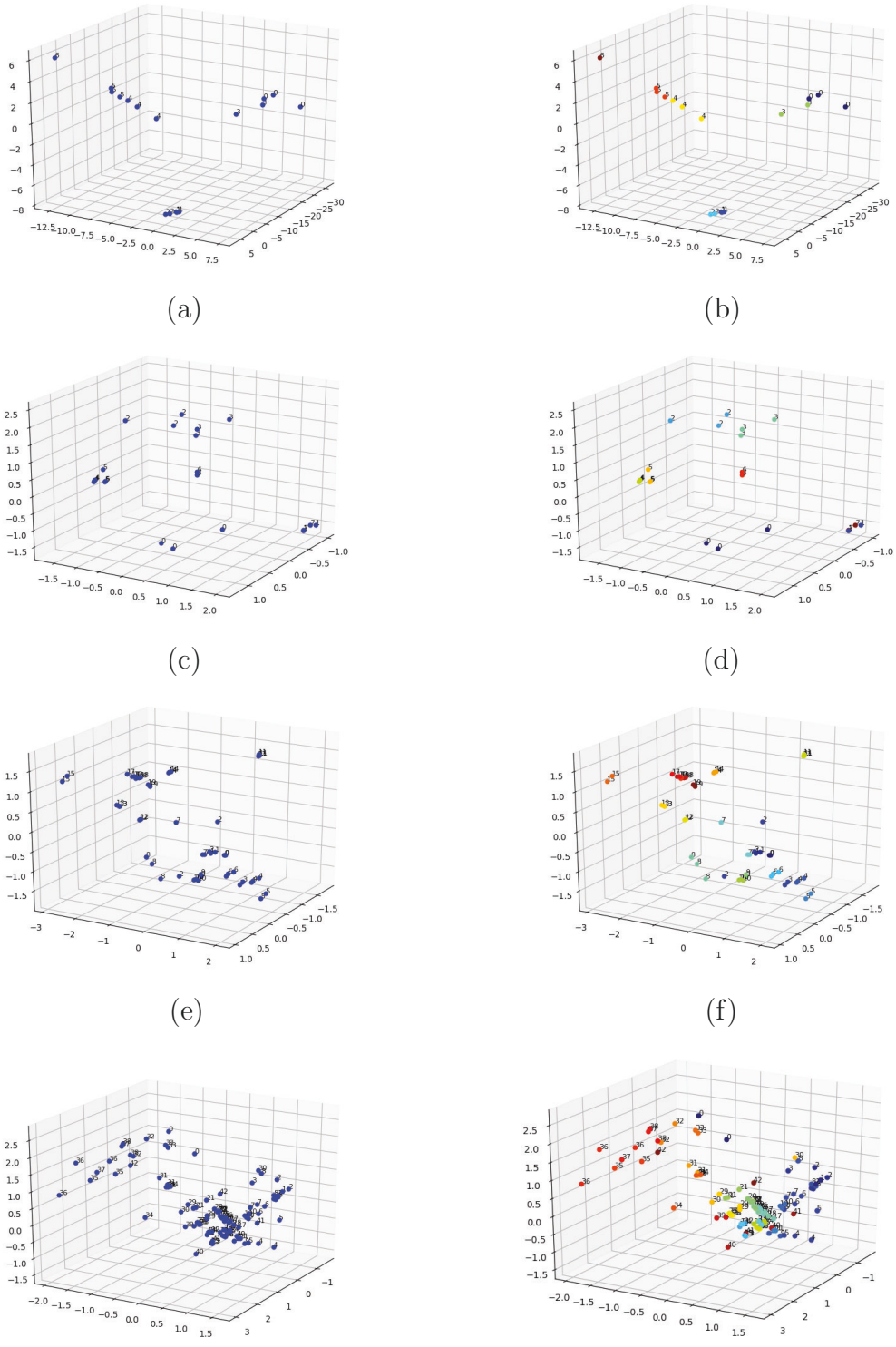
folds when the clustering approach MRC is used. In fact, PgRC1.2 achieves an average performance gain of 7.69% on the four databases when MRC is used. Specifically, the gains are respectively 8.63%, 4.22%, 5.37%, and 12.57%, revealing a positive correlation with the average file size in each database (database-17: 2.86GB, database-21: 6.46GB, database-50: 1.42GB, database-100: 1.97GB). For minicom, the improvement is 11.05%, 4.51%, 4.47%, 1.13% for the four databases and 5.29% on average.

The improvements on SPRING and FaStore are lower than the other two algorithms, which range from 0.78% to 2.68% on different databases. The main reason is that SPRING and FaStore also compress the identifier and quality score (i.e., compressing the FASTQ format files), as the compression performance of quality scores is mainly dependent on the rate of smoothed quality scores. The increase in the overlapping redundancy between reads provided by MRC does not contribute to the compression. How to convert *quality scores* in a reads set into a characteristic feature vector is a topic for investigation.

Overall, as shown in Table 4.6, our proposed MRC method outperforms over compressing reads datasets separately in all cases.

4.4.2 3D Visualization for the Clusters of Reads Data Sets after Transformation

By the feature extraction and selection of MRC, a reads set is converted into a vector with m dimensions. To assess the effectiveness of our clustering method, we apply Multi-Dimensional Scaling (Cox & Cox 2008) on the original feature spaces of database-17, database-21, database-50, database-100 (see Figure 4.1 a, c, e, g), in comparison with the feature vectors constructed by MRC (the cluster labels marked in Figure 4.1 b, d, f, h). It can be seen that the clusters from MRC matches the distance space computed by Multi-Dimensional Scaling, suggesting that MRC is effective for selecting subgroups of high similarity for compression of multiple reads datasets.



(a) (b) (c) (d) (e) (f) (g) (h)

Figure 4.1: 3D visualisation of the feature space after Multi-Dimensional Scaling from (a) database-17, (b) database-17 by clustering of MRC; (c) database-21, (d) database-21 by clustering of MRC; (e) database-50, (f) database-50 by clustering of MRC; (g) database-100, (h) database-100 by clustering of MRC

4.4.3 PCA Analysis on the Feature Vectors Converted from the Reads Datasets

Table 4.7: Minimizers of high importance in our PCA analysis

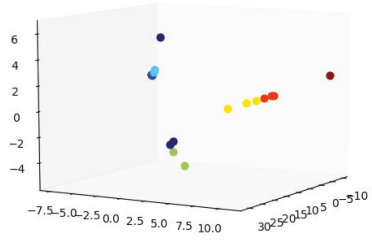
database-21			database-50			database-100		
v	w	minimizer	v	w	minimizer	v	w	minimizer
12	0.725	AAAAATA	15	0.725	AAAAATT	1	0.627	AAAAAAC
6	0.723	AAAAACG	22	0.694	AAAACCG	27	0.626	AAAACGT
4	0.710	AAAAACA	25	0.650	AAAACGC	25	0.581	AAAACGC
29	0.708	AAAACTC	27	0.646	AAAACGT	2	0.577	AAAAAAG
2	0.697	AAAAAAG	0	0.621	AAAAAAA	39	0.572	AAAAGCT
17	0.694	AAAACAC	38	0.614	AAAAGCG	13	0.563	AAAAATC
15	0.665	AAAAATT	24	0.607	AAAACGA	26	0.558	AAAACGG
18	0.664	AAAAAGT	11	0.601	AAAAAGT	38	0.552	AAAAGCG
0	0.657	AAAAAAA	37	0.599	AAAAGCC	20	0.551	AAAACCA
13	0.655	AAAAATC	21	0.577	AAAACCC	41	0.541	AAAAGGC

v indicates the mapped ID of a feature, w represents an importance score of a feature, a higher score indicates a more important feature.

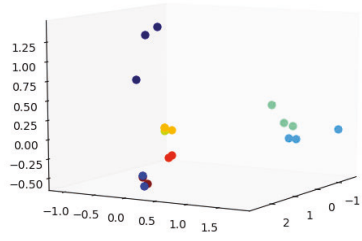
Our PCA analysis results are also displayed in Figure 4.2.

To get deeper insights into the roles of the minimizers in the clustering, Principal Component Analysis (Abdi & Williams 2010) is conducted on the feature vectors converted from each database to compare the importance of each feature. Principal Component Analysis (PCA) is a method to reduce the dimension of data by maximizing the variance of each dimension (Alpaydin 2020). By PCA, the original data (with dimension $n \times m$, each column f_j is a feature vector, $j \in [1, m]$) is transformed to an $n \times p$ matrix where $p \leq min(m, n)$, each column vector corresponds to one component $c_i = (c_{i1}, c_{i2}...c_{im}), i \in [1, p]$, c_i represents the direction of the i-th maximum variance of the original data, the absolute value of c_{ij} represents the magnitude of vector projection of feature f_j onto c_i, which can be considered

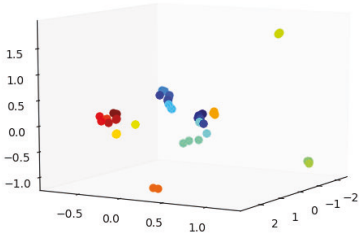
Figure 4.2: 3D Visualisation of the feature space after Principal Component Analysis from (a) database-17 (b) database-21; (c) database-50 (d) database-100



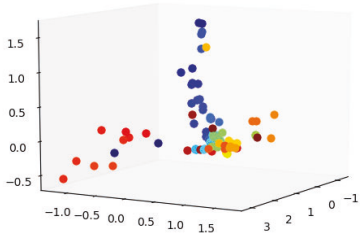
(a)



(b)



(c)



(d)

as the importance of feature j for c_i. Each c_i explains an amount of the variance of original matrix, the percentage of variance explained by c_i is denoted as vr_i.

The new components are efficient for data analysis but not interpretable enough. In order to understand which features are more important for clustering, we measure the importance of feature f_j through $w_j = \sum_{a=1}^{p} |c_{aj}| \times vr_a, j \in [1, m]$. The features with the highest importance for each database and the corresponding minimizers are listed in Table 4.7. Interestingly, some databases share common features. For example, the features with mapped IDs 0, 2, 15, 25, 27, and 38 occurred in more than one database, which can be considered as representative minimizers of these databases worth of further investigation.

4.4.4 Time Complexity and Speed Performance

Running time is an essential factor for measuring the applicability of an algorithm. For MRC, the time complexity of the compression depends on the compression algorithm. Here we present the time complexity for the transformation and feature vector construction. Let L and N_R denote the read length and total number of reads in a collection of reads set R. The computation of the minimizers of the reads takes $O(N_R(L - k + 1))$ time, where k is the length of the minimizers. The computation of the variance of each feature and sorting based on the value of variance takes $O(m_f n + m_f \log m_f)$ time, where m_f is the number of minimizers that appears at least one time in the datasets. In the worst case, $m_f = 4^k$, the time complexity is $O(4^k n + 2k4^k \log 2)$. The standard K-means clustering takes $O(tKn_e d)$, where t is the number of iterations, n_e is the number of vectors, d is the dimension of vectors. In the worst case (assuming K-means splits the original data into two clusters with size 1 and $n_e - 1$ each time), the clustering stage will apply $n - 3$ K-means with $K = 2, d = m, n_e = n, n - 1..., 4$, the time complexity is $O(2\sum_{n_e=4}^{n_e=n-3} td) = O((n + 1)(n - 3)tm)$. As $N_R L$ and 4^k is much larger than $n^2 m$ in normal cases, the running time of stages 1 and 2

Table 4.8: Transformation and clustering time of MRC (in second), red indicates the shortest compression time of each dataset.

Stage	database-17	database-21	database-50	database-100
Transformation+Clustering	149.54	492.98	321.16	850.63
Compression (by minicom)	1866.94	18682.44	5051.15	8786.49
Compression (by PgRC1.2)	1203.26	7543.40	4168.48	8565.72
Compression(by SPRING)	1936.84	5283.13	2325.19	11207.65
Compression(by FaStore)	2963.19	18870.07	15058.386	50333.904

are mainly related on N_R, L and k. When k is fixed, the running time has a positive linear relationship with $N_R L$, which is the total size of database. The actual speed of our method on different databases fits the analysis.

To achieve the performance of our algorithm in the compression of multiple reads datasets, we test the running time of MRC on the four databases. As mentioned before, the running time of MRC can be divided into two parts: transformation+clustering (only depending on the input databases) and compression (depending on both input set and compression algorithm).

MRC took 149.54, 492.98, 321.16, 850.63 seconds on the transformation and clustering stage for database-17, database-21, database-50 and database-100 respectively (Table 4.8), while the shortest compression time taken by the four algorithms on these databases are 1203.26, 5283.13, 2325.19, 8565.72 seconds. The transformation and clustering time is much shorter than the compression time.

4.5 Summary

In this work, we introduced MRC, a clustering-based algorithm for the compression of reads datasets. The key idea of this method is the measurement of the similarity between reads datasets based on minimizer frequency and the application of a variant K-means clustering to group the

reads datasets into subsets with a high similarity between each other. Reads datasets in each cluster are compressed together with additional information recorded to enable decompressing separately.

This algorithm is applicable for reads datasets with the same reads length. We have demonstrated that MRC achieved a substantial improvement in all cases compared to the original state-of-the-art compression algorithms. In addition, analysis of the time complexity on real databases shows that the time complexity of our algorithm is linear in practice, the time of additional preprocessing and clustering stage are much less than the compression time.

Availability

Supplementary information is available for this chapter at here. The C++ codes of our algorithm are available at https://github.com/ttan6729/MRC.

Chapter 5

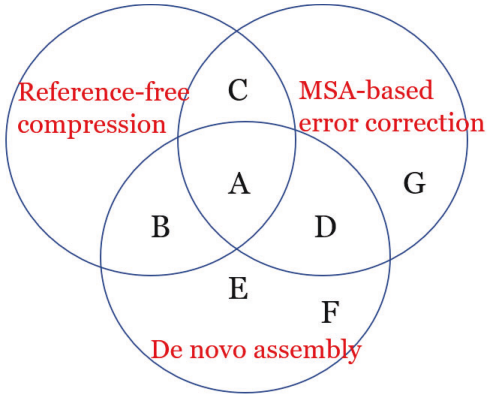
Assembly-improved Compression of Genomic Short Reads via Error Correction

5.1 Introduction

In this chapter, we focus on the relationship between de novo (reference-free) compression, error correction and de novo assembly of short reads data, especially the effect of error reduced input data on de novo compression.

As mentioned before, most of the methods for de novo compression, MSA based error correction and de novo assembly utilize overlapping between reads based on k-mers in different ways (see Figure 5.1). Multiple methods were designed for searching matched substring (overlaps) between sequence data (Liu, Wong & Li 2020). Based on their relationships, it is estimated that error correction can reduce the number of mismatched nucleotides without decreasing the quality of de novo assembly, therefore, we propose to do error correction before compression to improve the performance of reference-free compression.

Here we test the performance of state-of-the-art compression algorithms on the original and corrected data of four collections of reads sets, then



A. *k*-mer based and utilize overlaps between reads.
B. Merge overlapped reads into longer contigs.
C. Heuristic method for overlaps searching to reduce complexity.
D. Correct sequencing errors.
E. Compute all overlaps with length *k-1* in reads sets.
F. Correct errors based on special structures in de Bruijn Graph.
G. Correct errors based on alignment of reads

Figure 5.1: Common characteristics of reference-free compression, error correction and de novo assembly

compare the assembling results. To understand the compression performance on multiple reads sets, we proposed PMRC, a **P**ath graph based approach for **M**ultiple **R**eads Sets **C**ompression. The experiments show that high precision error correction method is able to reduce the number of mismatched nucleotides during compression, which increases the final compression ratio. We found that about 11-28% improvement in compression ratio is achieved on these databases. The same error correction also increases the quality of de novo assembly on most tested reads sets.

In the rest of this chapter, we comment on the relationships between these methods and introduce our method for the compression of multiple reads sets. We then present the detailed performance of de novo compression and de novo assembly results on the error-corrected data. We conclude that error correction is efficient for improving the quality of genome assemblies.

5.2 Error Correction and Compression

Reference-free compression tools normally construct a data structure for specific k-mers (e.g., prefix, suffix, other specific positions, minimizer (Roberts et al. 2004)) in each read, then search overlapping between reads using k-mer as seed, and merge overlapped reads into longer contigs, encode each read with regard to the contigs to achieve high compression ratio. Due to the characteristics of reference-free compression, compressing reads sets with overlaps together can increase the total number of matched subsequences other than compressing them separately, which will improve the final compression ratio. It can also be found that reference-free compression algorithms utilize the redundancy between reads data (Tang & Li 2020) in a similar way with MSA based error correction tools.

The performance of reference compression is seriously limited by the mismatched nucleotides between reads. When reads share common subsequence containing a few distinct nucleotides, the compression encoding costs more in comparison with when the mismatched nucleotides do not exist. In fact, mismatched nucleotides occur frequently in reference-free compression (e.g., 1513802 mismatched bases in the compression of SRR354183.fastq via minicom (Liu et al. 2019), a set of 5575465 reads with length 100). Extra space in the final encoding is required which decreases the compression ratio. For instance, as shown in the example below, read A can be encoded as "1 10", while read B needs to be encoded as "1 4 G 6 10".

Contig	G	A	A	C	T	C	A	T	A	C	G	T	C
Read A		A	A	C	T	C	A	T	A	C	G		
Read B		A	A	C	T	G	A	T	A	C	G		

Example of a consistent overlapping (A); and an overlapping containing a pair of mismatched bases (B).

Most state-of-the-art compression algorithms simply record the mismatched nucleotides for lossless compression, however, the reasons for occurrence of

mismatched bases are seldom explored.

Currently, reference-free compression focuses on short reads data (reads with length 36-200bp), which mainly suffers from substitution errors (Allam et al. 2015). Substitution errors are those bases that are wrongly called and hence substituted by other bases. Due to substitution error, the original sequence is replaced with the same sequence except a few bases.

MSA based error correction and reference-free compression share a similar strategy: merging subsets of reads generated by heuristic method (normally based on identical k-mer) into longer contigs, mismatched nucleotide will occur during that process. Then the reference-free compression methods record the mismatched nucleotide for lossless compression, while MSA based EC tools check each mismatched nucleotide and conduct correction on a part of them. Though the merging process has different goals: reference-free compression tends to find long overlapping to increase compression ratio and MSA based error correction keeps higher similarity in each subset to ensure the accuracy of correction. These two types of methods are complementary to each other, here we focus on the effect of MSA based error correction on reference compression, that is, high quality error correction can significantly reduce the number of mismatched nucleotides and hence improve the performance of compression.

5.3 Error Correction and Genome Assembly: Current Observations

In the previous section, we have demonstrated that MSA based error correction method can improve the performance of de novo compression. However, it is unclear that whether error correction will affect the quality of de novo assembly.

As mentioned before, de Bruijn graph is a preferred option for short reads assembly due to its great practicability in handling short reads data. Once the graph is constructed, longer sequences can be extracted from the path

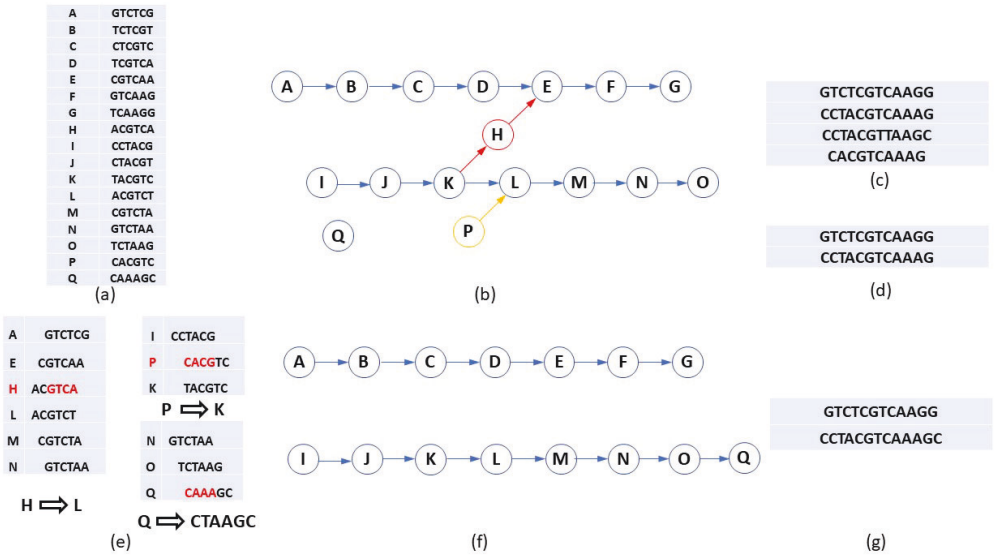


Figure 5.2: Example of de Bruijn based assembly of corrected and uncorrected data. (a) A set of 17 reads with potential errors. (b) The de-Bruijn graph based on reads set, red arrows indicates chimeric connection, yellow arrows indicates "tip" (nodes that disconnected on one end). (c) The assembly result based on original set, include redundant (the first three) and short (the last) contigs. (d) The assembly result based on original set after removing erroneous structures. (e) The MSA and corresponding correction, only the alignment of corrected reads are shown. (f) The de-Bruijn graph based on corrected reads set. Erroneous connection and nodes are removed. (g) The assembly result based on corrected set with continuous contigs.

directly.

However, short reads data always suffer from substitution errors, which complicates the construction: a de Bruijn graph is based on k-mer of each read, one single substitution error will lead to up to k erroneous nodes and corresponding edges in graph. These incorrect k-mers cause erroneous structures such as dead ends (tips), parallel paths (bubbles) and spurious connections (chimeric connections) (Heydari, Miclotte, Van de Peer & Fostier 2019) in graph, which result in more fragmented assembly.

In order to handle the errors in reads data, many assembly tools include built-in algorithms for error correction. A naive approach is to remove the low coverage nodes and the corresponding arcs, however, this can only correct genuine errors instead of the biological variants and randomly distributed errors (Zerbino & Birney 2008). In addition, important information may be lost during that. The error correction in state-of-the-art assembly tools are normally based on graph topology: identify the erroneous structures and remove the corresponding nodes.

Instead of removing the erroneous k-mers, MSA based EC tools identify erroneous k-mers based on the result of alignment and replace these k-mers with higher frequent ones via modifying a few nucleotides.

In most cases, the effect of correction on assembly is similar to removing erroneous k-mers: After correction, the nodes of original k-mers may disappear (if that k-mer no longer exists in reads set); as the corrected k-mers normally already exist in reads set, no new nodes will be generated. The difference between the de Bruijn graph based on corrected data and the one after removing the erroneous structures will be insignificant, and the corresponding assembly result will be similar.

One example is shown in Fig 5.2: for a set of 17 reads A-G, the constructed de Bruijn graph Fig 5.2b contains one chimeric connection (K→H→E) and one "tip" (P→L) which will be detected by built-in algorithms of de novo assembly and node H, P will be removed. In MSA based alignment (Fig 5.2d), read H is transformed to L and P is transformed to K, corrected reads

are mapped to existing nodes, Q is transformed to a unique sequence after correction and assembled to O. At last, the result of erroneous structures removing Fig 5.2(d) and MSA based error correction Fig 5.2(g) is only different in the last nucleotide of the second sequence.

5.4 Correction-integrated Method for Multiple Reads Sets Compression: An advancement for Both Data Compression and Genome Assembly

For a set of n reads sets $R = \{R_1, R_2...R_n\}$, denote the jth read of R_i as r_{ij}.

As the performance of reference-free compression mainly depends on the length of encoded overlaps between reads data, compressing the reads sets that have similar reads together can improve the overall compression ratio.

Here we propose an approach for selecting the to-be-compressed-together reads sets for compression of larger reads databases, which is named PMRC (Path graph based Multiple Reads sets Compression). PMRC consists of three stages: distance matrix construction, construct path graph, divide path graph into groups.

1. **Distance Matrix Construction**: Transform each R_i into a feature vector based on the normalized (w, k)-minimizer frequency of reads in R_i, construct an $n \times n$ distance matrix based on the feature vectors.

2. **Path graph Construction**: Construct a complete graph based on the distance between feature vectors, then construct path graph based on it.

3. **Divide path graph into groups**: Split path graph into groups via cutting off a part of edges.

5.4.1 Distance Matrix Construction

Minimizer is the lexicographically smallest k-mer in a sequence. The (w, k)-minimizer of a sequence is denoted as the k-minimizers of each $(w + k - 1)$ length subsequence in this sequence (Roberts et al. 2004), an example of $(4, 3)$-minimizer is shown as above. Minimizer is widely used in processing of reads data (Liu, Zhang, Zou & Zeng 2020).

Here for each reads set R_i, we compute the (w, k)-minimizers of reads that without 'N' letter, then count the frequency of each unique k-mer that occurs in all (w, k)-minimizers. Each minimizer s is mapped to a unique $2 \times k$ bit integers by $f(s) = \sum_{a=1}^{a=k} \phi(s[a] \cdot 4^{a-1}), \phi(A) = 0, \phi(C) = 1, \phi(G) = 2, \phi(T) = 3$. The original R_i is transformed into a feature vector $V_i = \{V_{i1}, V_{i2}..V_{i4^k}\}, V_{ij} = \frac{F(R_i,j)}{|R_i|}, F(R_i, j) =$ frequency of minimizer with value j in $(w, k)-$ minimizers of R_i. A set of n feature vectors is generated from n reads sets, for each feature $\{v_{1j}, v_{2j}..v_{nj}\}$, we denote the maximum and minimum values of this feature as v_{max}, v_{min}, each feature value v_{ij} of this feature is normalized by $norm(v_{ij}) = v_{min} + \frac{v_{ij}-v_{min}}{v_{max}-v_{min}}$, the normalized feature values range from 0 to 1. Then the m features with the highest variance are selected from normalized feature set as final feature space, denoted by $\underline{V} = \{\underline{V}_1, \underline{V}_2...\underline{V}_n\}, \underline{V_i} = \{\underline{V}_{i1}, \underline{V}_{i2}...\underline{V}_{im}\}$.

Then a distance matrix D with dimension $n \times n$ is computed as following:

$$D_{ij} = \begin{cases} 0 & i = j \\ \text{Euclidean distance between } S_i \text{ and } S_j & i \neq j \end{cases} \qquad i, j \in [1, n] \tag{5.1}$$

5.4.2 Path Graph Construction

Firstly, a complete graph (undirected graph in which every pair of distinct vertices is connected by a unique edge) with vertices $v_1, v_2..v_n$ is constructed based on the distance matrix D in the first stage, the edge between v_i, v_j is denoted as e_{ij}, the weight of e_{ij} is denoted by $w_{ij} = e^{-D[i][j]}, w_{ij} \in [0, 1]$.

sequences	A	T	C	C	T	G	A	T	G
subsequence	**A**	**T**	**C**	C	T	G			
with		T	**C**	**C**	**T**	G	A		
minimizers			**C**	**C**	**T**	G	A	T	
(in bold)				C	T	G	**A**	**T**	**G**

$(4, 3)$-minimizer of a length 9 sequence

A path graph is a graph with two degree 1 vertices and all other vertices are with degree 2, here we construct the path graph as following:

1. Define the set of all existing edges as E, set p as an array with length n, $w(i)$ indicates the highest weight among edges that connect node i in E.

2. Select the edge e_{ab} with maximum weight in E, remove e_{ab}, e_{ba} from E.

3. If $w(a) \leq (b)$, $p[1] = b, p[2] = a$, else $p[1] = b, p[2] = a$, set current vertex id $c = p[2]$, remove all edges that connect $v_{p[1]}$ from E, set i as 3.

4. Find the edge e_{cd} with weight $w(c)$, $p[i] = d$, increment i, remove all edges that connect v_c from E, set current vertex id $c = d$.

5. Repeat step 4 until i equals n.

The order of vertices in path graph are represented as p, an array of vertices.

5.4.3 Divide Path Graph into Groups

In the second stage, the vertices array p of path graph is generated. As the complexity of most compression algorithms exceeds $O(NL)$, where N and L are the read number and read length of reads set, to reduce the compression time, the total size of each to-be-compressed-together reads sets needs to be limited, in PMRC, it is achieved via dividing the sets into groups with limited size.

The division of path graph includes two steps: first, divide path graph into clusters with higher inter-similarity by cutting off edges with low weight, for vertices array p and the corresponding edges, the first part of division is as following:

1. Define vertices array cur_g as [ ], add $v_{p[1]}, v_{p[2]}$ to cur_g, define set of vertices array $G1 = \emptyset$, define c as 2.

2. Threshold $e = 0.9 - \frac{2}{|cur\_g|+5}$, $cur\_w = w_{p[c-1]p[c]}$, if $w_{p[c]p[c+1]} \geq e \cdot cur\_w$ or $w_{p[c]p[c+1]} \geq$ average weight of edges in path graph, jump to step 3, else jump to step 4.

3. Add $v_{p[c+1]}$ to cur_g, increment c.

4. Add cur_g to G_1, $cur\_g = [v_{p[c+1]}]$, increment c.

5. Repeat step 2 until c equals length of p.

In the second step, each set in $G1$ is divided into groups with size less than or equal to 3, define set of vertices set $G2 = \emptyset$, then for each group g in $G1$, the process is as following:

1. Define $cur\_g = \emptyset$, $c = 2$, add $g[1], g[2]$ to cur_g, $cur\_w = w_{g[1]g[2]}$.

2. $e = \frac{6}{10-|cur\_g|}$, if $|cur\_g| \leq 3$ and $w_{g[c]g[c+1]} \geq cur\_w \cdot e$, jump to step 3, else jump to step 4.

3. Add $g[c+1]$ to cur_g, increment c.

4. Add cur_g to $G2$, $cur\_g = \{g[c+1]\}$, increment c.

5. Repeat step 2 until $(|g| - c) \leq 2$.

6. If $|cur\_g| + (|g| - c) \leq 3$, add $g[c+1], g[c+2]$ to cur_g, then add cur_g to $G2$, else add cur_g and $\{g[c+1], g[c+2]\}$ to $G2$.

Then the corresponding reads sets of vectors in each group of $G2$ will be compressed together in order preserving mode. The read number and file name of each reads sets will be recorded for further decompression. In decompression process, the compressed file of each group will be decompressed, then each reads set is restored using the recorded read number.

5.5 Advancement: Error-corrected Compression of Reads Databases and their Assembly Performance

We test the performance of reference-free compression and de novo assembly on the data corrected by MSA based EC tools, the performance of our approach PRMC is also tested. We conduct experiments at a computing cluster running Red Hat Enterprise Linux 6.7 (64 bit) with 2 Intel Xeon E5-2695 processors (2.3GHz, 14Cores), 128 GB of RAM.

Four databases of reads sets were collected to test the effect of error correction on reference-free compression and de novo assembly. Details about the collections of reads data sets are shown in Table 5.1. The original data were downloaded from http://ftp.sra.ebi.ac.uk/vol1/fastq/. The URLs of each file are listed in Supplementary Data.

5.5.1 Compression Ratio Gain

Two state-of-the-art compression algorithms, minicom (Liu et al. 2019) and PgRC1.2 (Kowalski & Grabowski 2020*a*), were tested on four collections of reads sets to understand the performance improvement achieved by our method PMRC and MSA based error correction in comparison with compressing the uncorrected set separately via the same compression algorithm. Both algorithms were executed in order preserving mode, other parameters follow the default setting (e.g. 24 threads for minicom, 8 threads for PgRC1.2). The C++ codes of PMRC are freely available from

Table 5.1: Four collections of reads data sets

Collections	number of reads sets	total size (in GB)	length of reads
database-33	33	102.39	101
database-47	47	208.97	100
database-50	50	71.04	50
database-100	100	197.97	36

More information of each file can be found in Supplementary Data

https://github.com/ttan6729/PMRC.

Here to evaluate the improvement in compression ratio, we select Karect (Allam et al. 2015), which corrects the four selected datasets without change in the original size of each reads set, that is, the size of all corrected data in the table is exactly the same as original data.

The performance improvement is measured by performance *gain*, which is defined as:

$$gain = 1 - \frac{CS_1}{CS_2} \times 100\%$$

CS_1 = Compressed size with PMRC on corrected data.

CS_2 = Compressed size of one-by-one compression on uncorrected data.

A higher value of *gain* indicates more compression improvement.

Table 5.2 shows the compressed size of the same algorithms using uncorrected databases+one-by-one approach and corrected databases (by Karect (Allam et al. 2015))+PMRC approach.

It can be seen that Karect+PMRC achieved substantial improvement on uncorrected data with one-by-one approach: for PgRC1.2, the improvements

Table 5.2: Compressed file size (in byte) comparison between the straightforward *one-by-one* approach and error correction (via Karect) + PMRC approach

Collection	minicom			PgRC1.2		
	one-by-one	Karect+PMRC	*gain*	one-by-one	Karect+PMRC	*gain*
database-33	3370577920	2563737600	23.94%	3257922008	2355886080	27.69%
database-47	8162744320	6541660160	19.86%	8060349515	6239426560	22.59%
database-50	3714529280	3366123520	9.38%	3006844741	2683299840	10.76%
database-100	9593702400	8971356160	6.49%	8828937571	7807191040	11.57%

are 23.94%, 19.86%, 9.38%, 6.49% on databases-33, 47, 50, 100 respectively and 14.92% on average, for minicom, the improvements are 27.69%, 22.59%, 10.76%, 11.57% respectively and 18.15% on average. The improvements are more significant on databases with longer reads. In addition, the improvement on PgRC1.2 is a bit higher than minicom. The main reason is that PgRC1.2 requires overlaps without mismatched bases for high quality reads (reads that without 'N'), hence more sensitive to the corrected bases with no mismatch.

Overall, Karect+PMRC outperforms one-by-one approach without correction in all cases.

In order to gain deeper insight into the effect of error correction, we count the number of mismatched nucleotides of uncorrected and corrected (by Karect) reads set during the compression process of minicom. As shown in Table 5.3, the number of mismatched nucleotides of corrected data is much less than the uncorrected data in the same reads set.

The results in Table 5.2 and 5.3 verify our previous estimation: error correction will significantly reduce the number of mismatched nucleotides of reference-free compression and hence improve the compression ratio.

Reads set	Number of mismatched nucleotides	
	original	corrected
ERR532390_1	31049795	5946148
ERR532394_1	63378199	18256422
SRR067591	2091817	405840
SRR354180	1866568	72884
SRR354181	1396543	72058
SRR354182	1445706	60300
SRR354183	1513802	85801
SRR361240	9000536	2738571

Table 5.3: Number of mismatched nucleotides of minicom

5.5.2 Effect on De Novo Assembly

To assess the impact of error correction on de novo assembly results, the corrected and uncorrected reads sets from the four databases in Table 5.1 are assembled using SPAdes (Bankevich et al. 2012). SPAdes implements a built-in error correction procedure that removes erroneous k-mers through the identification of parallel paths ('bubbles' and 'tips') in de Bruijn graph. User can disable this procedure using "only-assembler" mode (by default this procedure is enabled). Here to understand the effect of MSA based error correction and the built-in error correction of de novo assembly tools, the uncorrected data is assembled in default and "only-assembly" mode respectively, the data corrected by Karect is assembled in "only-assembly" mode. The resulting assemblies were evaluated using QUAST (Gurevich, Saveliev, Vyahhi & Tesler 2013) and detailed reports are provided in the supplementary file.

Here we show the NGA50 values of each reads set in Table 5.4. The NGA50 value indicates the length of the assembled contigs which are considered as subsequences that contain no major structural assembly errors, a higher NGA50 value implies a less fragmented assembly and hence higher

Table 5.4: NGA50 of the assembled by SPAdes in different modes and with correction of Karect in advance

id	Name	SPAdes (only assembly)	SPAdes (correction and assembly)	Karect+SPAdes (only assembly)
1	SRR354182	108313	113327↑↑	110217↑
2	SRR361240	1570	1800 ↑↑	1950 ↑↑
3	SRR361242	4658	5170 ↑↑	5511 ↑↑
4	SRR361244	2554	2889 ↑↑	3087 ↑↑
5	SRR387478	95978	109436 ↑↑	109332 ↑↑
6	SRR387479	74802	75374 =	77924 ↑
7	SRR401414	1713	1684 ↓	1724 =
8	SRR401415	1842	1864=	1870 =

Symbols in this table are based on their value relative to the NGA50 value of uncorrected data ("only-assembly" mode) as follows: ↓↓-10% <↓< 0% < = < 3% <↑< 10%↑↑

quality. For reads sets 2 to 5, both the built-in correction and Karect have larger than 10% improvement. The improvements are larger than 3% for reads set 1 and less than 3% for reads set 8. Remarkably, for reads set 7, built-in correction leads to lower NGA50 value and Karect only has less than 3% improvement. For reads set 6, the improvement of built-in correction is less than 3%.

Compared with built-in correction of SPAdes, the NGA50 values of Karect are a bit higher on reads set 6,7,8 (3.38%, 2.38%, 0.32%) and show more significant improvements on reads set 2,3,4 (8.33%, 6.60%, 6.85%). For reads set 1 and 5, Karect slightly decrease the NGA50 values (-2.74%, -0.10%). Overall, MSA based error correction has a positive effect on de novo assembly in comparison with de novo assembly and its built-in error correction procedure.

Table 5.5: Computation time of PMRC and compression time of PgRC1.2 and minicom on each corrected database (in second).

Stage	database-33	database-47	database-50	database-100
PMRC	675.46	760.13	263.45	821.24
Compression (by minicom)	11935.08	23557.64	5051.15	8786.49
Compression (by PgRC1.2)	2870.7	9712.13	4168.48	8565.72

5.5.3 Complexity Analysis and Speed Performance

Running time is an essential factor for measuring the applicability of an algorithm. For PMRC, the computation time relates to the number of reads set n, total reads in the reads sets $|R|$, length of each read L (assume all the reads have the same length) and value of w, k, m. In the first stage, the computation of (w, k) minimizers of each read takes $O(|R|[(L-k+1)+k+(L-w+1)]) = O(|R|(2L-w+2))$ time (compute values of all k-mers first and then find k-mer with smallest value in $(L-w+1)$ subsequences, the selection of m features from 4^k unique k-mers requires $O(4^k n + 2k4^k \log 2)$, then the construction of distance matrix will take $O(\frac{(n^2-n)m}{2})$. In the second stage, $(n-1)$ steps will be performed with complexity $O(n), O(n-2)..O(2)$, the total time complexity is $O(\frac{n^2+n-2}{2})$. In the third stage, the time complexity is $O(I)$, where I is the number of iterations, in the worst case, $I = \frac{n}{2} - 1$ and complexity is $O(\frac{n}{2}) - 1$. Hence the total time complexity of PMRC is $O(|R|(2L-w+2)+4^k n+2k4^k \log 2+\frac{n^2(m+1)-n(m-2)}{2}-2)$, as $|R|(2L-w+2)$ and $4^k n$ are normally much larger than mn, the running time of PMRC mainly depends on $|R|L$ (the total size of reads sets) and $|R|w$.

Table 5.5 reports the running time of PMRC and compression time of two compression algorithms, it can be seen the additional computation time of PMRC is much shorter than the fastest compression algorithms on the same database.

5.6 Summary

We have shown that MSA based error correction can be successfully used in improving the performance of reference-free compression on short reads datasets: the corrected substitution errors increase the number of continuous overlaps between reads data, hence the number of mismatched nucleotides is reduced, which results in improvement in final compression ratio. In addition, compression of multiple reads sets can benefit from proper selection of to-be-compressed-together reads sets.

By comparing the assembling results of original data and corrected data, we show that error correction normally have a positive effect on de novo assembly, the state-of-the-art EC tools have a better performance than built-in error correction procedure of de novo assembly tools in terms of the quality of assembly.

Availability

The supplementary file can be downloaded from here. The C++ codes of our algorithm are available at https://github.com/ttan6729/PMRC.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Next-generation sequencing (NGS) technologies have generated terabytes of data over the past 15 years. The high-throuput NGS data has enabled a comprehensive analysis of genomes and dramatically accelerated biological and biomedical research. The wealth data also proposed computational challenges for data storage and analysis. Efficient compression tools for NGS data are crucially needed.

In this thesis, we introduce Efficient methods for the compression of large collections of genomes and reads datasets. We also demonstrate and verify the relationship between de novo assembly, error correction and reference-free compression.

In Chapter 3, we present ECC, a clustering-based reference selection algorithm for the reference-based compression of genome data. It measures the difference between genome sequences via MinHash distance, then clusters the genome sequences into subsets with higher inter-similarity than the original set. In experiments, ECC outperforms the previous one-by-one approach on compressing three datasets using six state-of-the-art compression algorithms. It achieved 2.22% to 22.83% compression ratio gain in different cases.

Then we introduce MRC as shown in Chapter 4. MRC is a clustering method for compressing multiple reads datasets. It transforms each file into a feature vector based on the k-minimizer frequency, then applies a variant of K-means clustering to determine the to-be-compressed-together files. We have shown that MRC obtains 4.22% to 12.57% compression ratio gain on 17-100 reads sets using state-of-the-art de novo compression algorithm. Moreover, the additional computation time is much shorter than the original compression time.

In Chapter 5, we explore the relationship between compression, error correction and de novo assembly of short reads data. We demonstrate that high quality error correction can improve the performance of reference-free compression via significantly reducing the number of mismatched nucleotides during compression. The experiments on four datasets verified our estimation. We also found that MSA based error correction has a positive effect on de novo assembly in most cases.

6.2 Future Work

There are some potential problems and research directions that relate to this thesis, which are summarized as following.

1. Although our method ECC provides an efficient reference selection scheme for reference-based compression, there are some other facets worth of investigation for further improvement. First, ECC is unable to handle dynamic genome sequence dataset. When new sequence is added to the compressed dataset, it can only be compressed with the final reference in previous compression. There are two possible ways to solve that: 1. Store the sketch set information of existing centroid sequences and update the clustering result based on new sequence. 2. Select the reference for new sequence via heuristic method. In addition, we did not exploit the structure of representative sequences of each dataset provided.

2. MRC used a variant K-means clustering method to divide reads sets into groups. In experiments, groups with only one element will occur, which may lose potentially suitable subgroups that to be compressed together. We implement a procedure to rebalance cluster size, however, it may affect the accuracy of the clustering result. We will investigate the methods that can naturally avoid single element group.

3. We prove that de novo compression can benefit from error reduced input data. However, this improvement requires the available error correction tools. How to design built-in error correction procedure for de novo compression tools is another challenging problem. As the primary purpose of de novo compression and error correction is different, the procedure needs to achieve a balance between reducing mismatched nucleotide during compression and keeping precision of error correction.

Bibliography

Abdi, H. & Williams, L. J. (2010), 'Principal component analysis', *Wiley interdisciplinary reviews: computational statistics* **2**(4), 433–459.

Alic, A., Tomás, A., Torres, J. S., Medina, I. & Blanquer, I. (2014), Robust error correction for de novo assembly via spectral partitioning and sequence alignment., *in* 'IWBBIO', pp. 1040–1048.

Allam, A., Kalnis, P. & Solovyev, V. (2015), 'Karect: accurate correction of substitution, insertion and deletion errors for next-generation sequencing data', *Bioinformatics* **31**(21), 3421–3428.

Alpaydin, E. (2020), *Introduction to machine learning*, MIT press.

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D. et al. (2012), 'Spades: a new genome assembly algorithm and its applications to single-cell sequencing', *Journal of computational biology* **19**(5), 455–477.

Bellman, R. (1966), 'Dynamic programming', *Science* **153**(3731), 34–37.

Berlin, K., Koren, S., Chin, C.-S., Drake, J. P., Landolin, J. M. & Phillippy, A. M. (2015), 'Assembling large genomes with single-molecule sequencing and locality-sensitive hashing', *Nature Biotechnology* **33**(6), 623.

Bonfield, J. K. & Mahoney, M. V. (2013), 'Compression of fastq and sam format sequencing data', *PloS one* **8**(3), e59190.

Bose, T., Mohammed, M. H., Dutta, A. & Mande, S. S. (2012), 'Bind–an algorithm for loss-less compression of nucleotide sequence data', *Journal of biosciences* **37**(4), 785–789.

Brandon, M. C., Wallace, D. C. & Baldi, P. (2009), 'Data structures and compression algorithms for genomic sequence data', *Bioinformatics* **25**(14), 1731–1738.

Broder, A. Z. (1997), On the resemblance and containment of documents, *in* 'Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)', IEEE, pp. 21–29.

Cánovas, R., Moffat, A. & Turpin, A. (2014), 'Lossy compression of quality scores in genomic data', *Bioinformatics* **30**(15), 2130–2136.

Cao, M. D., Dix, T. I., Allison, L. & Mears, C. (2007), A simple statistical algorithm for biological sequence compression, *in* '2007 Data Compression Conference (DCC'07)', IEEE, pp. 43–52.

Chandak, S., Tatwawadi, K., Ochoa, I., Hernaez, M. & Weissman, T. (2019), 'Spring: a next-generation compressor for fastq data', *Bioinformatics* **35**(15), 2674–2676.

Chandak, S., Tatwawadi, K. & Weissman, T. (2018), 'Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis', *Bioinformatics* **34**(4), 558–567.

Chen, X., Kwong, S. & Li, M. (2001), 'A compression algorithm for DNA sequences', *IEEE Engineering in Medicine and Biology Magazine* **20**(4), 61–66.

Chen, X., Li, M., Ma, B. & Tromp, J. (2002), 'DNACompress: fast and effective DNA sequence compression', *Bioinformatics* **18**(12), 1696–1698.

Cheng, K. O., Law, N. F. & Siu, W.-C. (2017), 'Clustering-based Compression for Population DNA Sequences', *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (1), 1–1.

Cheng, K.-O., Wu, P., Law, N.-F. & Siu, W.-C. (2015), 'Compression of Multiple DNA Sequences Using Intra-Sequence and Inter-Sequence Similarities', *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **12**(6), 1322–1332.

Chin, C.-S. & Khalak, A. (2019), 'Human genome assembly in 100 minutes', *bioRxiv* p. 705616.

Chiu, S. L. (1994), 'Fuzzy model identification based on cluster estimation', *Journal of Intelligent & fuzzy systems* **2**(3), 267–278.

Cleary, J. & Witten, I. (1984), 'Data compression using adaptive coding and partial string matching', *IEEE Transactions on Communications* **32**(4), 396–402.

Cock, P. J., Fields, C. J., Goto, N., Heuer, M. L. & Rice, P. M. (2009), 'The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants', *Nucleic Acids Research* **38**(6), 1767–1771.

Consortium, . G. P. et al. (2012), 'An integrated map of genetic variation from 1,092 human genomes', *Nature* **491**(7422), 56.

Cox, A. J., Farruggia, A., Gagie, T., Puglisi, S. J. & Sirén, J. (2016), RLZAP: Relative Lempel-Ziv with adaptive pointers, *in* 'International Symposium on String Processing and Information Retrieval', Springer, pp. 1–14.

Cox, M. A. & Cox, T. F. (2008), Multidimensional scaling, *in* 'Handbook of data visualization', Springer, pp. 315–347.

Danek, A., Deorowicz, S. & Grabowski, S. (2014), 'Indexes of large genome collections on a PC', *PloS One* **9**(10), e109384.

Deorowicz, S., Danek, A. & Grabowski, S. (2013), 'Genome compression: a novel approach for large collections', *Bioinformatics* **29**(20), 2572–2578.

Deorowicz, S., Danek, A. & Niemiec, M. (2015), 'GDC 2: Compression of large collections of genomes', *Scientific Reports* **5**, 11565.

Deorowicz, S. & Grabowski, S. (2011), 'Robust relative compression of genomes with random access', *Bioinformatics* **27**(21), 2979–2986.

Deorowicz, S. & Grabowski, S. (2013), 'Data compression for sequencing data', *Algorithms for Molecular Biology* **8**, 25.

Deorowicz, S., Kokot, M., Grabowski, S. & Debudaj-Grabysz, A. (2015), 'Kmc 2: fast and resource-frugal k-mer counting', *Bioinformatics* **31**(10), 1569–1576.

Dhanachandra, N., Manglem, K. & Chanu, Y. J. (2015), 'Image segmentation using k-means clustering algorithm and subtractive clustering algorithm', *Procedia Computer Science* **54**, 764–771.

Di Genova, A., Buena-Atienza, E., Ossowski, S. & Sagot, M.-F. (2019), 'Wengan: Efficient and high quality hybrid de novo assembly of human genomes', *bioRxiv* p. 840447.

Długosz, M. & Deorowicz, S. (2017), 'Reckoner: read error corrector based on kmc', *Bioinformatics* **33**(7), 1086–1089.

England, G. (2016), 'The 100,000 genomes project', *The* **100**, 0–2.

Garg, S., Fungtammasan, A. A., Carroll, A., Chou, M., Schmitt, A., Zhou, X., Mac, S., Peluso, P., Hatas, E., Ghurye, J. et al. (2019), 'Efficient

chromosome-scale haplotype-resolved assembly of human genomes', *bioRxiv* p. 810341.

Ginart, A. A., Hui, J., Zhu, K., Numanagić, I., Courtade, T. A., Sahinalp, S. C. & David, N. T. (2018), 'Optimal compressed representation of high throughput sequence data via light assembly', *Nature Communications* **9**(1), 566.

Goodwin, S., McPherson, J. D. & McCombie, W. R. (2016), 'Coming of age: ten years of next-generation sequencing technologies', *Nature Reviews Genetics* **17**(6), 333.

Grabowski, S., Deorowicz, S. & Roguski, Ł. (2014), 'Disk-based compression of data from genome sequencing', *Bioinformatics* **31**(9), 1389–1395.

Greenfield, P., Duesing, K., Papanicolaou, A. & Bauer, D. C. (2014), 'Blue: correcting sequencing errors using consensus and context', *Bioinformatics* **30**(19), 2723–2732.

Grumbach, S. & Tahi, F. (1993), Compression of DNA sequences, *in* 'Data Compression Conference, 1993. DCC'93.', IEEE, pp. 340–350.

Grumbach, S. & Tahi, F. (1994), 'A new challenge for compression algorithms: genetic sequences', *Information Processing & Management* **30**(6), 875–886.

Gurevich, A., Saveliev, V., Vyahhi, N. & Tesler, G. (2013), 'Quast: quality assessment tool for genome assemblies', *Bioinformatics* **29**(8), 1072–1075.

Hach, F., Numanagić, I., Alkan, C. & Sahinalp, S. C. (2012), 'Scalce: boosting sequence compression algorithms using locally consistent encoding', *Bioinformatics* **28**(23), 3051–3057.

Harismendy, O., Ng, P. C., Strausberg, R. L., Wang, X., Stockwell, T. B., Beeson, K. Y., Schork, N. J., Murray, S. S., Topol, E. J., Levy, S.

et al. (2009), 'Evaluation of next generation sequencing platforms for population targeted sequencing studies', *Genome biology* **10**(3), R32.

Hastie, T., Tibshirani, R. & Friedman, J. (2009), Unsupervised learning, *in* 'The elements of statistical learning', Springer, pp. 485–585.

Heo, Y., Ramachandran, A., Hwu, W.-M., Ma, J. & Chen, D. (2016), 'Bless 2: accurate, memory-efficient and fast error correction method', *Bioinformatics* **32**(15), 2369–2371.

Heo, Y., Wu, X.-L., Chen, D., Ma, J. & Hwu, W.-M. (2014), 'Bless: bloom filter-based error correction solution for high-throughput sequencing reads', *Bioinformatics* **30**(10), 1354–1362.

Heydari, M., Miclotte, G., Demeester, P., Van de Peer, Y. & Fostier, J. (2017), 'Evaluation of the impact of illumina error correction tools on de novo genome assembly', *BMC bioinformatics* **18**(1), 374.

Heydari, M., Miclotte, G., Van de Peer, Y. & Fostier, J. (2019), 'Illumina error correction near highly repetitive dna regions improves de novo genome assembly', *BMC bioinformatics* **20**(1), 298.

Huang, Y.-T. & Huang, Y.-W. (2017), 'An efficient error correction algorithm using fm-index', *BMC bioinformatics* **18**(1), 524.

Ilie, L., Fazayeli, F. & Ilie, S. (2011), 'Hitec: accurate error correction in high-throughput sequencing data', *Bioinformatics* **27**(3), 295–302.

Ilie, L. & Molnar, M. (2013), 'Racer: rapid and accurate correction of errors in reads', *Bioinformatics* **29**(19), 2490–2493.

Jain, A. K. (2010), 'Data clustering: 50 years beyond k-means', *Pattern recognition letters* **31**(8), 651–666.

Jones, D. C., Ruzzo, W. L., Peng, X. & Katze, M. G. (2012), 'Compression of next-generation sequencing reads aided by highly efficient de novo assembly', *Nucleic acids research* **40**(22), e171–e171.

Kingsford, C. & Patro, R. (2015), 'Reference-based compression of short-read sequences using path encoding', *Bioinformatics* **31**(12), 1920–1928.

Koboldt, D. C., Steinberg, K. M., Larson, D. E., Wilson, R. K. & Mardis, E. R. (2013), 'The next-generation sequencing revolution and its impact on genomics', *Cell* **155**(1), 27–38.

Kolmogorov, M., Yuan, J., Lin, Y. & Pevzner, P. A. (2019), 'Assembly of long, error-prone reads using repeat graphs', *Nature biotechnology* **37**(5), 540–546.

Koren, S., Walenz, B. P., Berlin, K., Miller, J. R., Bergman, N. H. & Phillippy, A. M. (2017), 'Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation', *Genome research* **27**(5), 722–736.

Korodi, G., Rissanen, J., Astola, J. et al. (2007), 'Dna sequence compression-based on the normalized maximum likelihood model', *IEEE Signal Processing Magazine* **24**(1), 47–53.

Kowalski, T., Grabowski, S. & Deorowicz, S. (2015), 'Indexing arbitrary-length k-mers in sequencing reads', *PloS one* **10**(7), e0133198.

Kowalski, T. & Grabowski, S. P. (2020*a*), 'Engineering the compression of sequencing reads', *bioRxiv* .

Kowalski, T. M. & Grabowski, S. (2020*b*), 'Pgrc: pseudogenome-based read compressor', *Bioinformatics* **36**(7), 2082–2089.

Kuruppu, S., Puglisi, S. J. & Zobel, J. (2011), Optimized relative Lempel-Ziv compression of genomes, *in* 'Proceedings of the Thirty-Fourth Australasian Computer Science Conference-Volume 113', Australian Computer Society, Inc., pp. 91–98.

Lawrence, M. S., Stojanov, P., Mermel, C. H., Robinson, J. T., Garraway, L. A., Golub, T. R., Meyerson, M., Gabriel, S. B., Lander, E. S. & Getz,

G. (2014), 'Discovery and saturation analysis of cancer genes across 21 tumour types', *Nature* **505**(7484), 495.

Li, H. (2016), 'Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences', *Bioinformatics* **32**(14), 2103–2110.

Li, R., Zhu, H., Ruan, J., Qian, W., Fang, X., Shi, Z., Li, Y., Li, S., Shan, G., Kristiansen, K. et al. (2010), 'De novo assembly of human genomes with massively parallel short read sequencing', *Genome research* **20**(2), 265–272.

Liu, Y., Peng, H., Wong, L. & Li, J. (2017), 'High-speed and high-ratio referential genome compression', *Bioinformatics* **33**(21), 3364–3372.

Liu, Y., Schröder, J. & Schmidt, B. (2013), 'Musket: a multistage k-mer spectrum-based error corrector for illumina sequence data', *Bioinformatics* **29**(3), 308–315.

Liu, Y., Wong, L. & Li, J. (2020), 'Allowing mutations in maximal matches boosts genome compression performance', *Bioinformatics* . btaa572.
URL: *https://doi.org/10.1093/bioinformatics/btaa572*

Liu, Y., Yu, Z., Dinger, M. E. & Li, J. (2019), 'Index suffix–prefix overlaps by (w, k)-minimizer to generate long contigs for reads compression', *Bioinformatics* **35**(12), 2066–2074.

Liu, Y., Zhang, X., Zou, Q. & Zeng, X. (2020), 'Minirmd: accurate and fast duplicate removal tool for short reads via multiple minimizers', *Bioinformatics* . btaa915.

Luo, R., Liu, B., Xie, Y., Li, Z., Huang, W., Yuan, J., He, G., Chen, Y., Pan, Q., Liu, Y. et al. (2012), 'Soapdenovo2: an empirically improved memory-efficient short-read de novo assembler', *Gigascience* **1**(1), 2047–217X.

Lupski, J. R., Reid, J. G., Gonzaga-Jauregui, C., Rio Deiros, D., Chen, D. C., Nazareth, L., Bainbridge, M., Dinh, H., Jing, C., Wheeler, D. A. et al. (2010), 'Whole-genome sequencing in a patient with charcot–marie–tooth neuropathy', *New England Journal of Medicine* **362**(13), 1181–1191.

MacQueen, J. et al. (1967), Some methods for classification and analysis of multivariate observations, *in* 'Proceedings of the fifth Berkeley symposium on mathematical statistics and probability', Vol. 1, Oakland, CA, USA, pp. 281–297.

Mardis, E. R. (2011), 'A decade's perspective on DNA sequencing technology', *Nature* **470**(7333), 198.

Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., Berka, J., Braverman, M. S., Chen, Y.-J., Chen, Z. et al. (2005), 'Genome sequencing in microfabricated high-density picolitre reactors', *Nature* **437**(7057), 376–380.

Matsumoto, T., Sadakane, K. & Imai, H. (2000), 'Biological sequence compression algorithms', *Genome Informatics* **11**, 43–52.

ENCODE Project Consortium (2012), 'An integrated encyclopedia of DNA elements in the human genome', *Nature* **489**(7414), 57.

Medvedev, P., Scott, E., Kakaradov, B. & Pevzner, P. (2011), 'Error correction of high-throughput sequencing datasets with non-uniform coverage', *Bioinformatics* **27**(13), i137–i141.

Miller, J. R., Koren, S. & Sutton, G. (2010), 'Assembly algorithms for next-generation sequencing data', *Genomics* **95**(6), 315–327.

Minoche, A. E., Dohm, J. C. & Himmelbauer, H. (2011), 'Evaluation of genomic high-throughput sequencing data generated on illumina hiseq and genome analyzer systems', *Genome biology* **12**(11), R112.

Mohammed, M. H., Dutta, A., Bose, T., Chadaram, S. & Mande, S. S. (2012), 'Deliminate—a fast and efficient method for loss-less compression of genomic sequences: sequence analysis', *Bioinformatics* **28**(19), 2527–2529.

Ochoa, I., Hernaez, M., Goldfeder, R., Weissman, T. & Ashley, E. (2016), 'Effect of lossy compression of quality scores on variant calling', *Briefings in Bioinformatics* **18**(2), 183–194.

Ochoa, I., Hernaez, M. & Weissman, T. (2014), 'iDoComp: a compression scheme for assembled genomes', *Bioinformatics* **31**(5), 626–633.

Ochoa, I., Hernaez, M. & Weissman, T. (2015), 'iDoComp: a compression scheme for assembled genomes', *Bioinformatics* **31**(5), 626–633.

Ondov, B. D., Treangen, T. J., Melsted, P., Mallonee, A. B., Bergman, N. H., Koren, S. & Phillippy, A. M. (2016), 'Mash: fast genome and metagenome distance estimation using minhash', *Genome Biology* **17**(1), 132.

Park, H.-S. & Jun, C.-H. (2009), 'A simple and fast algorithm for k-medoids clustering', *Expert Systems with Applications* **36**(2), 3336–3341.

Patro, R. & Kingsford, C. (2015), 'Data-dependent bucketing improves reference-free compression of sequencing reads', *Bioinformatics* **31**(17), 2770–2777.

Pavlichin, D. S., Weissman, T. & Yona, G. (2013), 'The human genome contracts again', *Bioinformatics* **29**(17), 2199–2202.

Pearson, W. R. & Lipman, D. J. (1988), 'Improved tools for biological sequence comparison', *Proceedings of the National Academy of Sciences* **85**(8), 2444–2448.

Pell, J., Hintze, A., Canino-Koning, R., Howe, A., Tiedje, J. M. & Brown, C. T. (2012), 'Scaling metagenome sequence assembly with probabilistic

de Bruijn graphs', *Proceedings of the National Academy of Sciences* **109**(33), 13272–13277.

Peng, Y., Leung, H. C., Yiu, S.-M. & Chin, F. Y. (2010), Idba–a practical iterative de bruijn graph de novo assembler, *in* 'Annual international conference on research in computational molecular biology', Springer, pp. 426–440.

Pevzner, P. A., Tang, H. & Waterman, M. S. (2001), 'An eulerian path approach to dna fragment assembly', *Proceedings of the national academy of sciences* **98**(17), 9748–9753.

Pratas, D., Pinho, A. J. & Ferreira, P. J. (2016), Efficient compression of genomic sequences, *in* '2016 Data Compression Conference (DCC)', IEEE, pp. 231–240.

Quail, M. A., Smith, M., Coupland, P., Otto, T. D., Harris, S. R., Connor, T. R., Bertoni, A., Swerdlow, H. P. & Gu, Y. (2012), 'A tale of three next generation sequencing platforms: comparison of ion torrent, pacific biosciences and illumina miseq sequencers', *BMC genomics* **13**(1), 1–13.

Rasheed, Z. & Rangwala, H. (2013), A map-reduce framework for clustering metagenomes, *in* '2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum', IEEE, pp. 549–558.

Roberts, M., Hayes, W., Hunt, B. R., Mount, S. M. & Yorke, J. A. (2004), 'Reducing storage requirements for biological sequence comparison', *Bioinformatics* **20**(18), 3363–3369.

Roguski, Ł., Ochoa, I., Hernaez, M. & Deorowicz, S. (2018), 'Fastore: a space-saving solution for raw sequencing data', *Bioinformatics* **34**(16), 2748–2756.

Saha, S. & Rajasekaran, S. (2015), 'ERGC: An efficient referential genome compression algorithm', *Bioinformatics* **31**(21), 3468–3475.

Saha, S. & Rajasekaran, S. (2016), 'NRGC: a novel referential genome compression algorithm', *Bioinformatics* **32**(22), 3505–3412.

Salomon, D. & Motta, G. (2010), *Handbook of data compression*, Springer Science & Business Media.

Sanger, F., Nicklen, S. & Coulson, A. R. (1977), 'Dna sequencing with chain-terminating inhibitors', *Proceedings of the national academy of sciences* **74**(12), 5463–5467.

Schulz, M. H., Weese, D., Holtgrewe, M., Dimitrova, V., Niu, S., Reinert, K. & Richard, H. (2014), 'Fiona: a parallel and automatic strategy for read error correction', *Bioinformatics* **30**(17), i356–i363.

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

Shendure, J. & Ji, H. (2008), 'Next-generation dna sequencing', *Nature biotechnology* **26**(10), 1135–1145.

Shi, W., Chen, J., Luo, M. & Chen, M. (2018), 'High efficiency referential genome compression algorithm', *Bioinformatics* .

Shibuya, Y. & Comin, M. (2019), 'Better quality score compression through sequence-based quality smoothing', *BMC bioinformatics* **20**(9), 1–11.

Shkarin, D. (2002), Ppm: One step to practicality, *in* 'Proceedings DCC 2002. Data Compression Conference', IEEE, pp. 202–211.

Simpson, J. T. & Durbin, R. (2012), 'Efficient de novo assembly of large genomes using compressed data structures', *Genome research* **22**(3), 549–556.

Siva, N. (2008), '1000 Genomes project', *Nature Biotechnology* **26**(1).

Song, L., Florea, L. & Langmead, B. (2014), 'Lighter: fast and memory-efficient sequencing error correction without counting', *Genome biology* **15**(11), 509.

Stein, J. C., Yu, Y., Copetti, D., Zwickl, D. J., Zhang, L., Zhang, C., Chougule, K., Gao, D., Iwata, A., Goicoechea, J. L. et al. (2018), 'Genomes of 13 domesticated and wild rice relatives highlight genetic conservation, turnover and innovation across the genus Oryza', *Nature Genetics* **50**(2), 285.

Tang, T. & Li, J. (2020), 'Transformation of fasta files into feature vectors for unsupervised compression of short reads databases', *Journal of Bioinformatics and Computational Biology* .

Tang, T., Liu, Y., Zhang, B., Su, B. & Li, J. (2019), 'Sketch distance-based clustering of chromosomes for large genome database compression', *BMC genomics* **20**(10), 1–9.

Tetko, I. V., Livingstone, D. J. & Luik, A. I. (1995), 'Neural network studies. 1. comparison of overfitting and overtraining', *Journal of chemical information and computer sciences* **35**(5), 826–833.

Treangen, T. J. & Salzberg, S. L. (2012), 'Repetitive dna and next-generation sequencing: computational challenges and solutions', *Nature Reviews Genetics* **13**(1), 36–46.

Von Luxburg, U. (2007), 'A tutorial on spectral clustering', *Statistics and computing* **17**(4), 395–416.

Wandelt, S., Bux, M. & Leser, U. (2014), 'Trends in genome compression', *Current Bioinformatics* **9**(3), 315–326.

Wandelt, S. & Leser, U. (2013), 'FRESCO: Referential compression of highly similar sequences', *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **10**(5), 1275–1288.

Wandelt, S. & Leser, U. (2015), 'MRCSI: compressing and searching string collections with multiple references', *Proceedings of the VLDB Endowment* **8**(5), 461–472.

Wang, W., Mauleon, R., Hu, Z., Chebotarov, D., Tai, S., Wu, Z., Li, M., Zheng, T., Fuentes, R. R., Zhang, F. et al. (2018), 'Genomic variation in 3,010 diverse accessions of Asian cultivated rice', *Nature* **557**(7703), 43.

Wenger, A. M., Peluso, P., Rowell, W. J., Chang, P.-C., Hall, R. J., Concepcion, G. T., Ebler, J., Fungtammasan, A., Kolesnikov, A., Olson, N. D. et al. (2019), 'Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome', *Nature biotechnology* **37**(10), 1155–1162.

Willems, F. M., Shtarkov, Y. M. & Tjalkens, T. J. (1995), 'The context-tree weighting method: basic properties', *IEEE transactions on information theory* **41**(3), 653–664.

Xie, X., Zhou, S. & Guan, J. (2015), 'CoGI: Towards compressing genomes as an image', *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **12**(6), 1275–1285.

Yager, R. R. & Filev, D. P. (1994), 'Generation of fuzzy rules by mountain clustering', *Journal of Intelligent & Fuzzy Systems* **2**(3), 209–219.

Yan, F., Lü, J., Zhang, B., Yuan, Z., Zhao, H., Huang, S., Wei, G., Mi, X., Zou, D., Xu, W. et al. (2018), 'The Chinese giant salamander exemplifies the hidden extinction of cryptic species', *Current Biology* **28**(10), R590–R592.

Yang, X., Dorman, K. S. & Aluru, S. (2010), 'Reptile: representative tiling for short read error correction', *Bioinformatics* **26**(20), 2526–2533.

Yanovsky, V. (2011), 'Recoil-an algorithm for compression of extremely large datasets of dna data', *Algorithms for Molecular Biology* **6**(1), 23.

Yu, Y. W., Yorukoglu, D., Peng, J. & Berger, B. (2015), 'Quality score compression improves genotyping accuracy', *Nature biotechnology* **33**(3), 240–243.

Zerbino, D. R. & Birney, E. (2008), 'Velvet: algorithms for de novo short read assembly using de bruijn graphs', *Genome research* **18**(5), 821–829.

Zhang, Y., Li, L., Yang, Y., Yang, X., He, S. & Zhu, Z. (2015), 'Light-weight reference-based compression of fastq data', *BMC bioinformatics* **16**(1), 188.

Zhao, L., Chen, Q., Li, W., Jiang, P., Wong, L. & Li, J. (2017), 'Mapreduce for accurate error correction of next-generation sequencing data', *Bioinformatics* **33**(23), 3844–3851.

Zhao, L., Xie, J., Bai, L., Chen, W., Wang, M., Zhang, Z., Wang, Y., Zhao, Z. & Li, J. (2018), 'Mining statistically-solid k-mers for accurate ngs error correction', *BMC genomics* **19**(10), 912.

Zhu, Z., Zhang, Y., Ji, Z., He, S. & Yang, X. (2013), 'High-throughput DNA sequence data compression', *Briefings in Bioinformatics* **16**(1), 1–15.

Zhu, Z., Zhang, Y., Ji, Z., He, S. & Yang, X. (2015), 'High-throughput dna sequence data compression', *Briefings in bioinformatics* **16**(1), 1–15.

Ziv, J. & Lempel, A. (1977), 'A universal algorithm for sequential data compression', *IEEE Transactions on information theory* **23**(3), 337–343.

Ziv, J. & Lempel, A. (1978), 'Compression of individual sequences via variable-rate coding', *IEEE Transactions on Information Theory* **24**(5), 530–536.