

University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

Computer Science and Engineering: Theses,
Dissertations, and Student Research

Computer Science and Engineering, Department
of

Fall 12-4-2020

SUFFIX TREE, MINWISE HASHING AND STREAMING ALGORITHMS FOR BIG DATA ANALYSIS IN BIOINFORMATICS

Sairam Behera

University of Nebraska-Lincoln, srbehera@gmail.com

Follow this and additional works at: <https://digitalcommons.unl.edu/computerscidiss>



Part of the [Computer Engineering Commons](#), and the [Computer Sciences Commons](#)

Behera, Sairam, "SUFFIX TREE, MINWISE HASHING AND STREAMING ALGORITHMS FOR BIG DATA ANALYSIS IN BIOINFORMATICS" (2020). *Computer Science and Engineering: Theses, Dissertations, and Student Research*. 201.

<https://digitalcommons.unl.edu/computerscidiss/201>

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in Computer Science and Engineering: Theses, Dissertations, and Student Research by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

SUFFIX TREE, MINWISE HASHING AND STREAMING ALGORITHMS FOR
BIG DATA ANALYSIS IN BIOINFORMATICS

by

Sairam Behera

A DISSERTATION

Presented to the Faculty of
The Graduate College at the University of Nebraska
In Partial Fulfilment of Requirements
For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professor Jitender S. Deogun

Lincoln, Nebraska

December, 2020

SUFFIX TREE, MINWISE HASHING AND STREAMING ALGORITHMS FOR
BIG DATA ANALYSIS IN BIOINFORMATICS

Sairam Behera, Ph.D.

University of Nebraska, 2020

Adviser: Jitender S. Deogun

In this dissertation, we worked on several algorithmic problems in bioinformatics using mainly three approaches: (a) a streaming model, (b) suffix-tree based indexing, and (c) minwise-hashing (minhash) and locality-sensitive hashing (LSH). The streaming models are useful for large data problems where a good approximation needs to be achieved with limited space usage. We developed an approximation algorithm (Kmer-Estimate) using the streaming approach to obtain a better estimation of the frequency of k -mer counts. A k -mer, a subsequence of length k , plays an important role in many bioinformatics analyses such as genome distance estimation. We also developed new methods that use suffix tree, a trie data structure, for alignment-free, non-pairwise algorithms for a conserved non-coding sequence (CNS) identification problem. We provided two different algorithms: STAG-CNS to identify exact-matched CNSs and DiCE to identify CNSs with mismatches. Using our algorithms, CNSs among various grass species were identified. A different approach was employed for identification of longer CNSs (≥ 100 bp, mostly found in animals). In our new method (MinCNE), the minhash approach was used to estimate the Jaccard similarity. Using also LSH, k -mers extracted from genomic sequences were clustered and CNSs were identified. Another new algorithm (MinIsoClust) that also uses minhash and LSH techniques was developed for an isoform clustering problem. Isoforms are generated from the same gene but by alternative splicing. As the isoform sequences share some exons but

in different combinations, regular sequencing clustering methods do not work well. Our algorithm generates clusters for isoform sequences based on their shared minhash signatures. Finally, we discuss *de novo* transcriptome assembly algorithms and how to improve the assembly accuracy using ensemble approaches. First, we did a comprehensive performance analysis on different transcriptome assemblers using simulated benchmark datasets. Then, we developed a new ensemble approach (Minsemble) for the *de novo* transcriptome assembly problem that integrates isoform-clustering using minhash technique to identify potentially correct transcripts from various *de novo* transcriptome assemblers. Minsemble identified more correctly assembled transcripts as well as genes compared to other *de novo* and ensemble methods.

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my advisor Dr Jitender S. Deogun for his support and guidance in the last few years here at UNL. We spent a lot of time discussing the ideas and possible solutions for various problems. I am really grateful to him for all his mentorship during my PhD research.

Besides my advisor, I would like to thank my committee members: Dr. Vinod N. Variyam, Dr. Lisong Xu, Dr. James C. Schnable and Dr. Etsuko N. Moriyama. It has been a matter of great pleasure and fortune to get a chance to collaborate and work with Dr. Schnable, Dr. Vinod and Dr. Moriyama for different projects. I thank Dr. Schnable for introducing me to the bioinformatics problems in plant science research. I also thank Dr. Vinod for helping me to understand some big data algorithms. My sincere thanks to Dr. Xu for his insightful comments on some of my research and his advice on my research progress.

I would like to thank Dr. Moriyama for allowing me to join her research lab at School of Biological Sciences. Without her support and guidance, it would not be possible to conduct this research. I am grateful to Computer Science and Engineering Department, Center for Root and Rhizobiome Innovation (CRRI) and Nebraska EPSCoR for supporting my research.

I thank my fellow lab-mates including Suraj, Baset, Natasha, Mohammed and Aziza for all their support and insightful discussions. I thank all professors, staffs and friends in CSE department for their help and encouragement throughout my PhD research. I am so grateful to members of Schnable lab for helping me to understand many concepts of biology and plant science. My sincere thank goes to Sutanu of Vinod's group, Xianjun and Zhikai from Schnable Lab and Adam and Kushagra from Moriyama lab for their help and collaboration.

Last but not the least, I would like to thank my family: my parents, parents-in-law, brothers, sisters, my wife and my beautiful daughter Saisha for supporting me throughout my PhD and my life in general.

PREFACE

- Some of the results presented in Chapter 2 have been published in [13].
- The results presented in Chapter 3 have been published in [74] and [11].
- The results presented in Chapter 4 will appear in [14].
- The results presented in Chapter 5 have been published in [15].
- Results presented in Chapter 6 have been published in [12], [145], and [17] and are included in a manuscript currently in preparation [16].

GRANT INFORMATION

This work has been partly supported by NSF EPSCoR RII Track-1: Center for Root and Rhizobiome Innovation (CRRI) Award OIA-1557417 to Etsuko N. Moriyama.

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of these agencies.

Table of Contents

List of Figures	xii
List of Tables	xiv
1 Introduction	1
2 Streaming algorithm for approximating k-mer frequency counts	5
2.1 Introduction	5
2.1.1 Problem statement	8
2.1.2 Related works	8
2.2 Methods	10
2.2.1 Implementation	14
2.3 Results	15
2.3.1 Experimental setup	15
2.3.2 Accuracy	18
2.3.3 Time and space	19
2.3.4 Sample size	21
2.4 Conclusion	22
3 Discovery of conserved non-coding sequences efficiently	24
3.1 Introduction	24

3.2	Background and Related Works	27
3.3	Methodology	30
3.3.1	Problem definition	31
3.3.2	Algorithm	31
3.3.3	CNS with mismatches	33
3.4	Experimental Results	36
3.4.1	Accuracy and sensitivity of our approach	37
3.4.2	Comparison of results from our approach and CDP	38
3.4.3	Association of CNSs with DNase hypersensitive sites	40
3.4.4	Running time	41
3.5	Conclusion and Future Works	42
4	Identifying conserved non-coding elements using min-wise hashing	44
4.1	Introduction	44
4.2	Materials and Methods	46
4.2.1	Minhash signatures	46
4.2.2	LSH-based clustering	49
4.2.3	CNE identification	52
4.2.4	Benchmark dataset	53
4.2.5	Performance evaluation	54
4.3	Results and Discussion	55
4.3.1	CNE identification performance	55
4.3.2	Time and space usage	57
4.4	Conclusion	59
5	Isoform clustering using minhash and locality-sensitive hashing	60
5.1	Introduction	60

5.2	Materials and Methods	64
5.2.1	Sequence comparison using minhash signatures	64
5.2.2	MinIsoClust isoform-clustering strategy	66
5.2.3	LSH-based bucketing	67
5.2.4	Identification of isoforms by clustering	68
5.2.5	Benchmark datasets	69
5.2.6	Performance evaluation	70
5.2.7	Program execution	72
5.3	Results and Discussion	72
5.3.1	Isoform-clustering accuracy	72
5.3.2	Computational time and space usage	74
5.4	Conclusion	75
6	New ensemble approach for improving transcriptome assembly	77
6.1	Introduction	78
6.2	Transcriptome Assembly Strategies	79
6.2.1	Genome-guided approach	80
6.2.2	<i>De novo</i> approach	81
6.2.3	Ensemble approach	83
6.3	Performance Evaluation of Transcriptome Assembly	84
6.3.1	Performance metrics without references	85
6.3.2	Performance metrics using actual biological data	86
6.3.3	Performance metrics using simulated benchmark data	87
6.4	Simulated Benchmark Transcriptome Datasets Generation	89
6.4.1	RNA-seq simulation methods	90
6.4.2	Examples of RNA-seq simulation	90

6.5	Performance Comparison among Transcriptome Assemblers	92
6.5.1	Genome-guided approach	93
6.5.2	<i>De novo</i> approach	95
6.5.3	Combining <i>de novo</i> assemblies generated using different <i>k</i> -mers	98
6.5.4	Analysis of <i>k</i> -mers used in assembled contigs	98
6.5.5	Ensemble approach	100
6.6	Minsemble: a New Ensemble Approach	102
6.6.1	Minhash signature generation	103
6.6.2	Clustering of potential isoforms	107
6.6.3	Selection of contigs for final assembly	108
6.6.4	Minsemble transcriptome assembly pipeline	110
6.6.5	Assembly performance evaluation	111
6.6.6	Results and discussion	113
6.6.6.1	Performance of transcriptome assembly at the tran- script level	113
6.6.6.2	Performance of transcriptome assembly at the gene level	118
6.6.6.3	Performance of transcriptome assembly for the single- isoform genes	121
6.6.6.4	Performance of transcriptome assembly for the multiple- isoform genes	121
6.7	Conclusion	124
7	Conclusion and future works	126
	Bibliography	127
	Appendix A	147

List of Figures

2.1	Flow-chart of KmerEstimate	11
2.2	Processing of streaming elements using hashmaps	13
2.3	k -mer count histogram for Human Chromosome 14 reads	16
2.4	k -mer count histogram for <i>Bombus impatiens</i> reads	16
2.5	k -mer count histogram for NA19238 reads	17
2.6	k -mer count histogram for HG004 reads	17
2.7	k -mer count histogram for HG14 reads	19
2.8	k -mer count histogram for Bumblebee reads	20
2.9	k -mer count histogram for NA19238 reads	21
2.10	k -mer count histogram for HG004 reads	22
3.1	Intersecting, overlapping, and independent MEMs	29
3.2	Algorithm for identifying CNSs	32
3.3	MEMs and weighted directed acyclic graph	34
3.4	Ranking of CNSs	35
3.5	Algorithm for finding exact-matched and mismatched CNSs	36
3.6	True positive discovery rate	38
3.7	CNSs identification analysis for multiple species	39
3.8	CNSs and DHSs for rice seedings and callus	41
3.9	Overlap rate of CNSs and DHSs in rice seedings and callus	41

4.1	Flowchart of MinCNE	48
4.2	Time and space usage of MinCNE and CNEFinder	58
5.1	Various alternative-splicing events	64
5.2	Flowchat of MinIsoClust	65
6.1	Contigs shared among genome-guided assemblers	96
6.2	Contigs shared among <i>de novo</i> assemblers (default)	97
6.3	Contigs shared among <i>de novo</i> assemblers (pooled)	99
6.4	performance comparison among different assembly methods	101
6.5	Minsemble procedure	103
6.6	Minsemble pipeline	104
6.7	Sequence similarity estimation using q -grams	105
6.8	Isoform and q -gram distribution	106
6.9	Retention of highly similar contigs and potential isoforms	109
6.10	Transcriptome assembler performance at the transcript level	114
6.11	Comparison of ensemble assemblers for correctly assembled contigs . . .	116
6.12	Comparison of ensemble assemblers for incorrectly assembled contigs . .	117
6.13	Transcriptome assembler performance at the gene level	120
6.14	Transcriptome assembler performance for the single-isoform genes	122
6.15	Transcriptome assembler performance for the multiple-isoform genes . . .	123
A.1	Comparison of ensemble assemblers for the Human dataset	156

List of Tables

2.1	Dataset specification	18
2.2	Accuracy of algorithms in estimating F_0 and f_1 for HG14 reads	18
2.3	Accuracy of algorithms in estimating F_0 and f_1 for Bumblebee reads	19
2.4	Accuracy of algorithms in estimating F_0 and f_1 for NA19238 reads	20
2.5	Accuracy of algorithms in estimating F_0 and f_1 for HG004 reads	21
3.1	Summary of CNS distribution	39
4.1	Generation of minhash signatures for two k -mer sequences S_1 and S_2	50
4.2	Comparison of MinCNE and CNEFinder	56
5.1	Distribution of numbers of isoforms in the four datasets	68
5.2	Isoform clustering performance among the four methods	69
5.3	Performance evaluation of isoform clustering using ARI	72
5.4	Number of singleton clusters generated by four methods	73
5.5	Run-time comparison among the four methods ^a	74
5.6	Space usage comparison among the four methods ^a	74
6.1	Comparison of transcriptome assembly performance among different methods	94
6.2	The k -mer analysis for the <i>de novo</i> assemblies using the <i>Z. mays</i> B73 and Rice datasets. ^a	100

6.3	Gene identification performance of transcriptome assemblers for “all genes”	118
6.4	Gene identification performance of transcriptome assemblers for the single- isoform genes	119
6.5	Gene identification performance of transcriptome assemblers for multiple- isoform genes	119
A.1	Isoform distribution	147
A.2	Transcriptome assembly performance for the <i>A. thaliana</i> No0 dataset at the transcript level	148
A.3	Transcriptome assembly performance for the <i>A. thaliana</i> Col0 dataset at the transcript level	148
A.4	Transcriptome assembly performance for the Rice dataset at the transcript level	148
A.5	Transcriptome assembly performance for the Soybean dataset at the tran- script level	149
A.6	Transcriptome assembly performance for the <i>Z. mays</i> B73 dataset at the transcript level	149
A.7	Transcriptome assembly performance for the <i>Z. mays</i> Mo17 dataset at the transcript level	149
A.8	Transcriptome assembly performance for the Human dataset at the tran- script level	150
A.9	Transcriptome assembly performance for the <i>A. thaliana</i> Col0 dataset at the gene level	150
A.10	Transcriptome assembly performance for the Human dataset at the gene level	150
A.11	Transcriptome assembly performance for the Rice dataset at the gene level	151

A.12 Transcriptome assembly performance for the Soybean dataset at the gene level	151
A.13 Transcriptome assembly performance for the <i>Z. mays</i> B73 dataset at the gene level	151
A.14 Transcriptome assembly performance for the <i>Z. mays</i> Mo17 dataset at the gene level	152
A.15 Transcriptome assembly performance for the <i>A. thaliana</i> Col0 dataset for the single-isoform genes	152
A.16 Transcriptome assembly performance for the Human dataset for the single-isoform genes	152
A.17 Transcriptome assembly performance for the Rice dataset for the single-isoform genes	153
A.18 Transcriptome assembly performance for the Soybean dataset for the single-isoform genes	153
A.19 Transcriptome assembly performance for the <i>Z. mays</i> B73 dataset for the single-isoform genes	153
A.20 Transcriptome assembly performance for the <i>Z. mays</i> Mo17 dataset for the single-isoform genes	154
A.21 Transcriptome assembly performance for the <i>A. thaliana</i> Col0 dataset for the multiple-isoform genes	154
A.22 Transcriptome assembly performance for the Human dataset for the multiple-isoform genes	154
A.23 Transcriptome assembly performance for the Rice dataset for the multiple-isoform genes	155
A.24 Transcriptome assembly performance for the Soybean dataset for the multiple-isoform genes	155

A.25 Transcriptome assembly performance for the <i>Z. mays</i> B73 dataset for the multiple-isoform genes	155
A.26 Transcriptome assembly performance for the <i>Z. mays</i> Mo17 dataset for the multiple-isoform genes	156

Chapter 1

Introduction

Computational biology or bioinformatics is an area of interdisciplinary research at the intersection of Computer Science and Biology that studies the methods of storing, retrieving and analyzing biological data i.e., sequences of nucleic acids (deoxyribonucleic acid “DNA” or ribonucleic acid “RNA”) and amino acids. The research in bioinformatics has been revolutionized in the last few decades by the advent of high-throughput technologies such as whole-genome sequencing and transcriptome sequencing. Traditional sequencing approaches have been replaced by next-generation and third-generation sequencing technologies that produce millions of sequences concurrently. The massive amount of sequencing data generated from these sequencing technologies poses a significant challenge for efficient and accurate analysis and interpretation of these large datasets. Currently used algorithms are often either not suitable for current data or not robust enough to deal with these large-scale datasets. More efficient methods and strategies are needed for better results.

One of the fundamental research area in bioinformatics is the analysis of nucleotide or protein sequences to understand their features, functions, and structures. This has brought a wide range of computational problems such as sequence alignment, sequence assembly, gene prediction, protein structure prediction, phylogenetic recon-

struction, and motif finding. With the availability of high-throughput next-generation sequencing (NGS) technologies, the field of sequence analysis requires efficient data structures and algorithms for large-scale data analysis. From a computational point of view, biological sequences (e.g., DNA, RNA, and amino acid sequences) can be regarded as strings that consist of a finite set of letters (comprising of individual four-letter alphabets for DNA and RNA, and 20-letter alphabets for proteins). The efficient processing of these strings or sequences is necessary for almost all sequence analysis techniques that are applied to bioinformatics data. Such techniques make extensive use of several string algorithms and succinct data structures such as suffix trees, suffix arrays, graphs, hashing methodologies, and sketching techniques.

In this dissertation research, we were interested developing new algorithms and approaches using efficient data structures and approaches such as streaming algorithm, suffix tree, sketching techniques, and efficient hashing methods. We developed new efficient algorithms or strategies for the following problems: (a) k -mer count frequency estimation, (b) conserved non-coding sequence discovery, (c) clustering of isoforms, and (d) improvement of *de novo* transcriptome assembly using a new ensemble approach.

The k -mers, i.e., subsequences of length k , play an important role in many bioinformatics problems including genome and transcriptome assembly. The counting of the occurrences of all k -mers and their frequency estimation is central steps in many large-scale sequence analyses such as metagenome analysis. In Chapter 2, we proposed an efficient hashing-based technique and streaming algorithm for approximating k -mer count frequencies in sequencing data.

The variable-length k -mers that are shared among sequences are representations of conserved regions in sequences. The conserved regions that do not code for proteins are found to play an important role in regulating gene expression. These regions are

called conserved non-coding sequences (CNSs) in plant science research or conserved non-coding elements (CNEs) in animal science research. The CNS (or CNE) discovery is important for various studies in comparative genomics. The existing pair-wise alignment methods are not scalable and efficient when multiple sequences are used at once. In Chapter 3, we proposed an alignment-free technique using a succinct data structure, e.g., suffix tree, for the discovery of exact matched CNSs and then we extended that approach by including a brute-force approach to find CNSs with few mismatches.

The CNSs in plants (15~50 bp) are much smaller than those in animals ($\geq 100bp$). Therefore, different approaches need to be adopted for identifying CNSs in plants and animals. In Chapter 4, we developed a better and efficient method for identifying longer CNS using minwise hashing (minhash) and locality-sensitive hashing (LSH) based techniques. As the conserved regions are shared among sequences, our approaches for CNS identification adopted a clustering technique to gather similar k -mers.

The clustering approaches are useful in many other bioinformatics applications. Some of the clustering techniques that we used for the CNS identification problem can also be reused for similar problems. In Chapter 5, we developed a new algorithm for the isoform clustering problem. The sequence similarity-based clustering methods usually generate false negatives when used for isoform clustering. Our algorithm performs clustering using shared minhash signatures that are generated due to shared exon regions in isoforms.

In bioinformatics, sequence assembly refers to aligning and merging short fragments of DNA sequences in order to reconstruct the original transcript sequences. Most of the existing *de novo* assembly algorithms misses many true original sequences. In Chapter 6, we studied various transcriptome assembly algorithms using simulated

dataset and then proposed an ensemble strategy using the clustering method that improves the recovery of isoforms.

Finally, in Chapter 7, we provide the conclusion of our dissertation research and discuss some of the remaining future works.

Chapter 2

Streaming algorithm for approximating k -mer frequency counts

Publication:

- Sairam Behera, Sutanu Gayen, Jitender S. Deogun, and N. V. Vinodchandran, 2018, “KmerEstimate: A Streaming Algorithm for Estimating k -mer Counts with Optimal Space Usage”, *In Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB '18)*, Washington DC, USA, pp. 438–447. DOI: <https://doi.org/10.1145/3233547.3233587>

2.1 Introduction

Counting distinct number of k -mers (substrings of length k in a DNA/RNA sequence), and more generally computing the frequency distribution of k -mers (k -mer abundance histogram), in a genome data is a central component of many methods in bioinformatics including sequence assembly [50, 79, 153, 114, 25], read error correction [2, 150], genome size prediction and estimation of its characteristics [61, 30, 83, 82], changes in copy number of highly repetitive sequences [84], digital normalization, [154, 110], and parameter tuning of k -mer analysis based tools [27, 28].

The next generation sequencing (NGS) technologies produce large amount of data that can be used to infer the genomes using *de novo* assembly algorithms. Approaches based on *de Bruijn* graphs [32] have been widely popular for both genome and transcriptome assembly. In this approach, sequencing reads are broken into smaller fragments of fixed size, i.e., k -mers. The distribution of k -mer frequency from sequencing data can be used to estimate the genomic characteristics such as genome size, repeat structure and heterozygous rate [83]. The sequencing reads always contain some erroneous bases due to the errors in sequencing process. In sequencing studies [95], it has been shown that in modern sequencing methods (such as high coverage sequencing) the majority of singleton k -mers, k -mers with frequency one, do not come from the genome, but are generated from sequencing errors and the number of erroneous k -mers is always much much larger than non-erroneous k -mers. But the frequency of these erroneous k -mers is very small because of high coverage used during sequencing. These erroneous k -mers can lead to mis-assembly when used in *de Bruijn* graph based assembly algorithms. Therefore, it is essential to either remove or correct the erroneous k -mers before using them in the assembly process. A first step towards this is to compute the number of k -mers with low frequency.

Several approaches such as sorting, suffix-array, efficient hashing, bloom filter, count-min sketch, burst trie, and parallel disk-based partitioning have been studied for computing exact number of distinct k -mers and in general for computing number of k -mers with a given frequency (for example frequency one). Various tools available based on the above approaches include Tallymer [73], Jellyfish [94], BFCOUNTER [96], DSK [115], MSPKmerCounter [81], KAnalyze [6], Khmer [154], KMC2 and KMC3 [36, 69], MSPKmerCounter [81], Gerbil [42], KCMBT [90], Turtle [118] and Squeakr [108]. However, the common problem with exact count methods is that it is not feasible to use for large datasets due to their inherent inefficiency based on time

and space complexity. Most of the existing computing resources could easily see their memory capacities used up due to high number of distinct k -mers that can be expected from the input.

A more practical approach is to not seek exact counts, instead to compute approximate counts that can be used to construct an approximate k -mer abundance histogram. Recently methods from data streaming algorithms have been proved to be very effective for approximating k -mer abundance histogram and related counts because of their very efficient memory and time usage. To date, the tools that use data streaming approach for k -mer counting problems are KmerGenie [28], KmerStream [95], ntCard [98], and Kmerlight [126].

We present KmerEstimate, a streaming algorithm that approximates the number of k -mers with a given frequency in a genomic data set. Our algorithm is based on a well known adaptive sampling based streaming algorithm for approximating distinct elements in a data stream due to Bar-Yossef, Jayram, Kumar, Sivakumar, and Trevisan [8]. In our implementation, the accuracy of the results are within 0.6% error rate and in general better than existing frequency count estimation algorithms. The time and space efficiency of our implementation of the algorithm is comparable to that of *ntCard*, the best known streaming approach known so far. In addition, our algorithm has provable approximation and space and time usage guarantees. We also show lower bounds on the space usage of any algorithm that approximates frequency distribution (one lower bound for additive approximation and another one for multiplicative approximation). The additive approximation lower bound implies that our algorithm is space optimal up to a polylogarithmic factor.

2.1.1 Problem statement

In data streaming model, the input is a stream of data items s_1, s_2, \dots, s_m where each s_i is coming from a known universe of items. The goal of a streaming algorithm is to approximate a function of interest in an on-line manner: the algorithm can only store limited amount of information and the computation is done in a single pass over the stream. This model has been very popular in designing algorithms for massive data set problems. The main resources of interest are the space usage, which is desired to be much smaller than the size of the stream, and the processing times per data item. We refer the reader to [99] for an overview and survey of data stream algorithms. In our application, each data item is a k -mer, a sub-sequence of k consecutive bases. For a k -mer κ , the *frequency* of κ is the number of occurrences of κ in the data set. Let f_i be the number of k -mers with frequency i . Thus f_1, f_2, \dots, f_m denotes the number of k -mers with frequency $1, 2, \dots, m$ respectively. We are interested in approximating f_i for all i (which is known as the k -mer abundance histogram of the data set). We also denote by \mathcal{S} the set of distinct items in the stream (the set of all distinct k -mers) and by F_0 the number of distinct items in the stream, thus $F_0 = |\mathcal{S}|$.

2.1.2 Related works

KmerGenie [28] is the first tool that uses a streaming approach to generate abundance histograms of k -mers. To estimate the best k -value for *de Bruijn* graph based assembly, kmerGenie generates the abundance histogram for different k -values and then analyze them to select the optimum k -mer size. The basic intuition behind this approach is that the optimum value of k should be the one that produces fewest erroneous k -mers. As compared to the exact count algorithms, it has demonstrated to be an order of magnitude faster, while providing nearly accurate results. It uses the idea

explored in [33] to create an approximate histogram by sampling from the k -mers. The hash function $\rho_\epsilon : \{A, C, T, G\}^k \rightarrow [0, \epsilon]$ uniformly distributes the universe of all possible k -mers into ϵ buckets. Then it counts the abundances of only those k -mers that hash to 0. The abundance histogram is then computed from the k -mer counts, scaling the number of k -mers with a given abundance by ϵ .

KmerStream [95] is another algorithm that estimate frequency statistics of k -mers using streaming approach. The authors of KmerStream focus on computing f_1 . To estimate f_1 , they adapt an approach of Bar-Yossef et al [8] (a different algorithm from [8] than the one our algorithm is based on). They also give a theoretical guarantee on their algorithm’s performance. In particular, for estimating f_1 of a data stream, they give an algorithm with the performance guarantee stated in the next theorem (for comparison sake, we state a stronger version of their result than that is given in their paper)¹.

[[98]] There is a streaming algorithm that, on a data stream over n items, outputs an estimate \hat{f}_1 for f_1 such that $|\hat{f}_1 - f_1| \leq \epsilon F_0$ with probability at least $\frac{2}{3}$, where F_0 is the number of distinct items in the data stream. It uses $O(\frac{1}{\epsilon^2} \log n)$ space and $O(1)$ update time.²

Kmerlight [126] uses the approach from KmerStream to compute estimates of k -mer abundance histogram (i.e. f_i for all i).

The most recent addition to the list of streaming approaches for k -mer abundance histogram estimation is ntCard [95]. It also uses a hash-based approach similar to KmerStream [98]. For efficient implementation ntCard takes advantage of ntHash [97] algorithm to compute the canonical hash values of k -mers. The s upper bits of

¹Melsted and Halldórsson gave a proof in the supplementary materials and had to assume hash functions are perfectly random for their analysis.

²It is assumed that any $O(\log m)$ bits operation, such as, computing and comparing each hash value takes $O(1)$ time, where m is the length of the stream. See [8, 66, 95]

64 bit hash value used for sampling the k -mers, picking the k -mers having at least s leading zeros. The leading r bits are used to build a frequency table of size 2^r for sampled k -mers. The experimental results of ntCard has higher accuracy rates than the existing approaches, using similar amount of memory. We note that none of these existing algorithms, other than the f_1 estimator due to Melsted and Halldórsson [98], have any theoretical guarantees on their performance. The complexity of our algorithm together with the lower bound on the additive approximation implies that our algorithm is provably space optimal upto small polylogarithmic factors.

2.2 Methods

Our KmerEstimate algorithm for estimating f_i for any $i > 0$, is a slightly modified version of Algorithm 1. A schematic of our algorithm is given in Figure 2.1. The basic idea is to sample a set of k -mers from \mathcal{S} , the set of distinct k -mers appearing in the stream. This is done by hashing each k -mer uniformly at random to 64 bits. Then, we only keep those k -mers which has rightmost s bits all zeros, for some s as specified below. This amounts to sampling rate $1/2^s$. Let B_s be the set of sampled k -mers. For each sampled k -mer, we also count it's frequency in the stream. Up to this point, our algorithm is same as ntCard, but the next operations are different and arguably simpler.

After the entire stream is processed, we compute the number k_i , the number of sampled k -mers which have frequency exactly i in the stream. Our estimate is $\hat{f}_i = k_i \cdot 2^s$.

Similar to ntCard algorithm, our algorithm samples k -mers with number of trailing zeros $\geq s$ in the corresponding 64 bit hash value. However, unlike *ntCard*, our algorithm chooses the value of s adaptively. The value of s is fixed to either 7 or 11

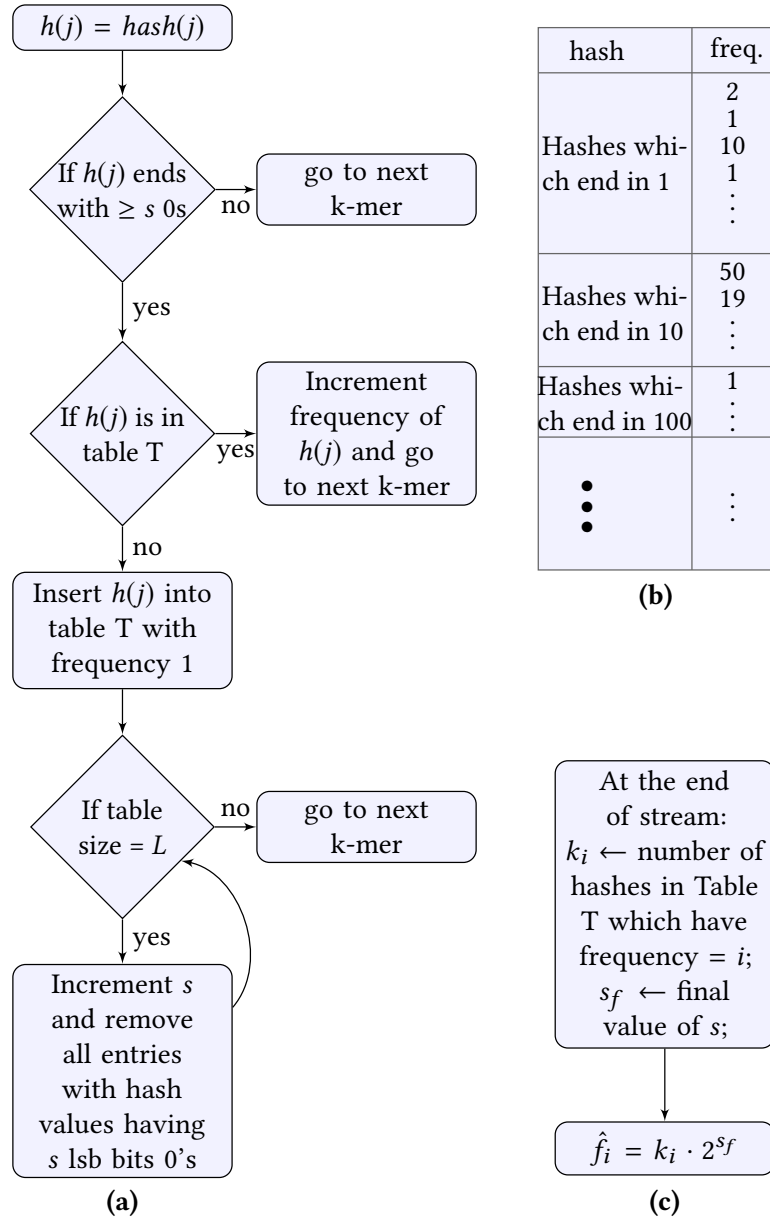


Figure 2.1: (a) The update process for the current k-mer j . The variable s is initialized to 0. (b) Table T used by the update and output process. T is partitioned into 65 hashmaps based on the number of trailing zeros. A maximum of $(L - 1)$ entries are allowed in T, for a parameter L . (c) The estimate for f_i for any $i > 1$, at the end of stream. The details are in Section 2.2.

for ntCard depending on the input size. We constrain the sample size $< L$ based on our desired approximation factor. We start with a sampling rate of $s = 0$, and as soon as the sample size becomes L , we double the sampling rate by making $s = 1$. It is simple to update the already sampled items. All those samples which have at least 1 trailing zeros in their 64-bit hash value are retained. Those samples which have hash value ending with a 1 are discarded. We keep on increasing s in this manner if needed until the entire stream is processed. Let the final value of s be s_f . This is the sampling rate of our algorithm. Thus we do not need to fix a sampling rate beforehand.

Algorithm 1: Algorithm for estimating f_i

```

1  $h \leftarrow$  a 2-wise independent uniformly random hash function mapping
    $[n] \rightarrow [n]$ ;
2  $g \leftarrow$  a 2-wise independent uniformly random hash function mapping
    $[n] \rightarrow [\Theta(\frac{1}{\epsilon^4} \log^2 n)]$ ;
3  $L \leftarrow$  a parameter fixed in the analysis;
4  $B_0 \leftarrow \phi$ ;
5  $s \leftarrow 0$ ;
6 for arrival of data item  $j \in \{1, 2, \dots, n\}$  in the stream do
7   if  $\text{zeros}(h(j)) \geq s$  then
8     if  $\text{key} = (g(j), \text{zeros}(h(j))) \in B_s$  then
9        $\text{key.count} = \text{key.count} + 1$ ;
10    else
11       $\text{Insert}(\text{key}, B_s)$ ;
12       $\text{key.count} = 1$ ;
13    end
14  end
15  while  $|B_s| \geq L$  do
16     $B_{s+1} \leftarrow$  Remove all keys  $(\alpha, \beta)$  with  $\beta = s$  from  $B_s$  ;
17     $s \leftarrow s + 1$ ;
18  end
19 end
   /* At the end of stream */
20  $s_f \leftarrow s$ ; // final value of  $s$ 
21  $k_i \leftarrow$  the number of samples in  $B_{s_f}$  with count value exactly  $i$ ;
22 Return  $\hat{f}_i = k_i \cdot 2^{s_f}$ ;

```

Algorithm 1 uses double hashing as a space-saving trick . The idea is that if the required sample size is small, instead of storing the entire hash value of each k -mer, a smaller hash of the first hash (double-hash) of the k -mer is enough for distinctly identifying each unique k -mer with high probability. But in our implementation, we did not use this double hashing. Instead we found the following approach gives satisfactory results. For storing the sampled k -mers in a multiplicity table that avoids collision, we used 65 space-efficient hashmaps instead of a single array. Each hashmap stores sampled k -mers with certain number of trailing zeros. For example, the sampled k -mers with exactly 3 trailing zeros stored in 3^{rd} hashmap (as shown in Fig. 2.2). Our program starts with $s = 0$ and once the total number of sampled k -mers reaches the sample size, it deletes s^{th} hashmap and increments s value. This process is continued until it finishes reading of sequences. The size of sampled k -mers is fixed in our algorithm and is given as an input parameter. The total number of sampled k -mers are always less than or equal to the sample size. Use of array of 65 hashmaps, avoids high error rates that is caused due to collisions.

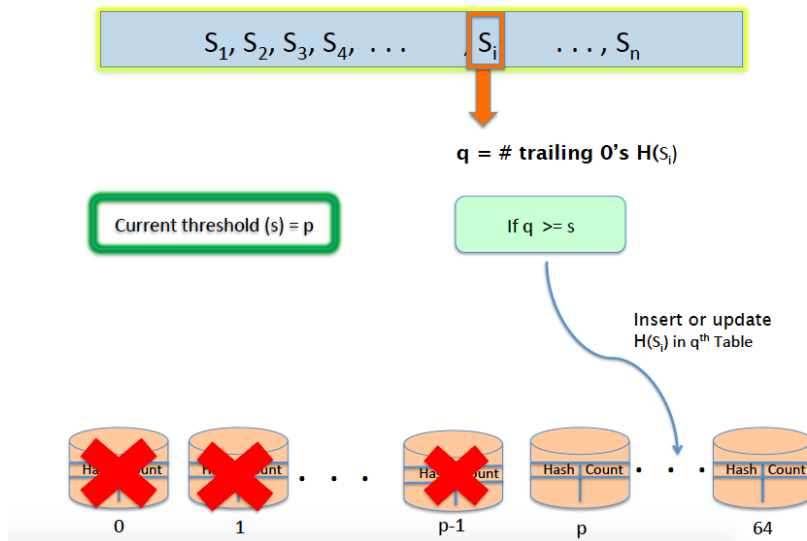


Figure 2.2: Processing of streaming elements using hashmaps

We used *nthash* [97] for computing 64 bit hash values of canonical k -mers that does not contain non-ACGT characters. The default hashmaps in C++ like `map` or `unordered_map` are not space efficient, so we used `sparsepp` [106] for storing sampled k -mers. Our program is written in C++ and distributed under GNU Public License (GPL). The current implementation does not support multi-threading. As input, it gets fasta/fastq file, size of the k -mer and sample size. The source codes and relevant documents including additional results are freely available at <https://github.com/srbehera11/kmerEstimate>.

2.2.1 Implementation

Similar to *ntCard* algorithm, our algorithm samples k -mers with number of trailing zeros $\geq s$ in the corresponding 64 bit hash value. However, unlike *ntCard*, our algorithm choses the value of s adaptively. The value of s is fixed to either 7 or 11 for *ntCard* depending on the input size. For storing the sampled k -mers in a multiplicity table that avoids collision, we used 64 space-efficient hashmaps instead of a single array. Each hashmap stores sampled k -mers with certain number of trailing zeros. For example, the sampled k -mers with exactly 3 trailing zeros stored in 3^{rd} hashmap. Our program starts with $s = 1$ and once the total number of sampled k -mers reaches the sample size, it deletes s^{th} hashmap and increments s value. This process is continued until it finishes reading of sequences. The size of sampled k -mers is fixed in our algorithm and is given as an input parameter. The total number of sampled k -mers are always less than or equal to the sample size. Use of array of 65 hashmaps, avoids high error rates that is caused due to collisions.

We used *nthash* [97] for computing 64 bit hash values of canonical k -mers that does not contain non-ACGT characters. The default hashmaps in C++ like `map` or `unordered_map` are not space efficient, so we used `sparsepp` [106] for storing sam-

pled k -mers. Our program is written in C++ and distributed under GNU Public License (GPL). The current implementation does not support multi-threading. As input, it gets fasta/fastq file, size of the k -mer and sample size. The source codes and relevant documents are freely available at <https://github.com/srbehera11/kmerEstimate>.

2.3 Results

2.3.1 Experimental setup

For evaluating the performance and accuracy of `kmerEstimate`, we used following publicly available sequencing datasets. The first two dataset are used in `KmerStream` and the last two datasets are used in `ntCard`. The information of all datasets are given in Table 2.1.

- 2x101 bp Human Chromosome 14 from Genome Assembly Gold-standard Evaluation (GAGE) [120]
- 2x124 bp *Bombus impatiens* (bumblebee) from GAGE dataset
- 500 bp Homo Sapiens dataset from the 1000 Genomes Project, for the individual NA19238 (SRA:ERR309932) [132]
- 2x250 bp paired-end Illumina whole genome shotgun sequencing data for the Ashkenazi mother (HG004) from The Genome in a Bottle (GIAB) project.[158]

We evaluated the performance of `kmerEstimate` by comparing with `KmerStream`, and `ntCard`. The accuracy of the results are compared against the result of `DSK`, the exact k -mer counting tool. The results were obtained on a single Core of Xeon E5-2697 v4 2.3GHz server.

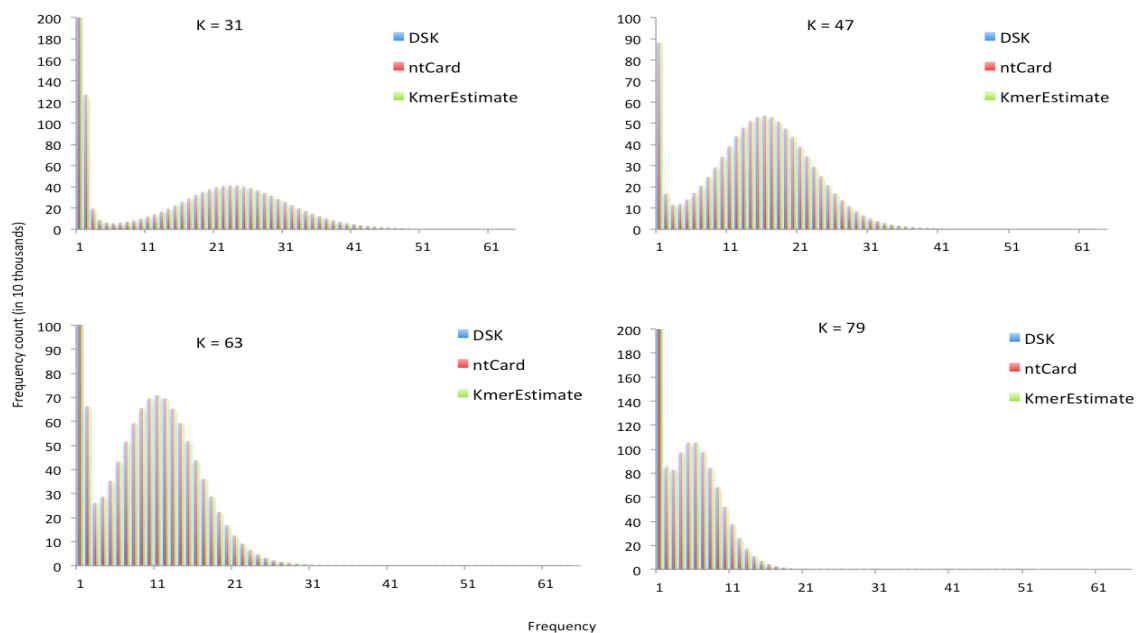


Figure 2.3: k -mer count histogram for Human Chromosome 14 reads

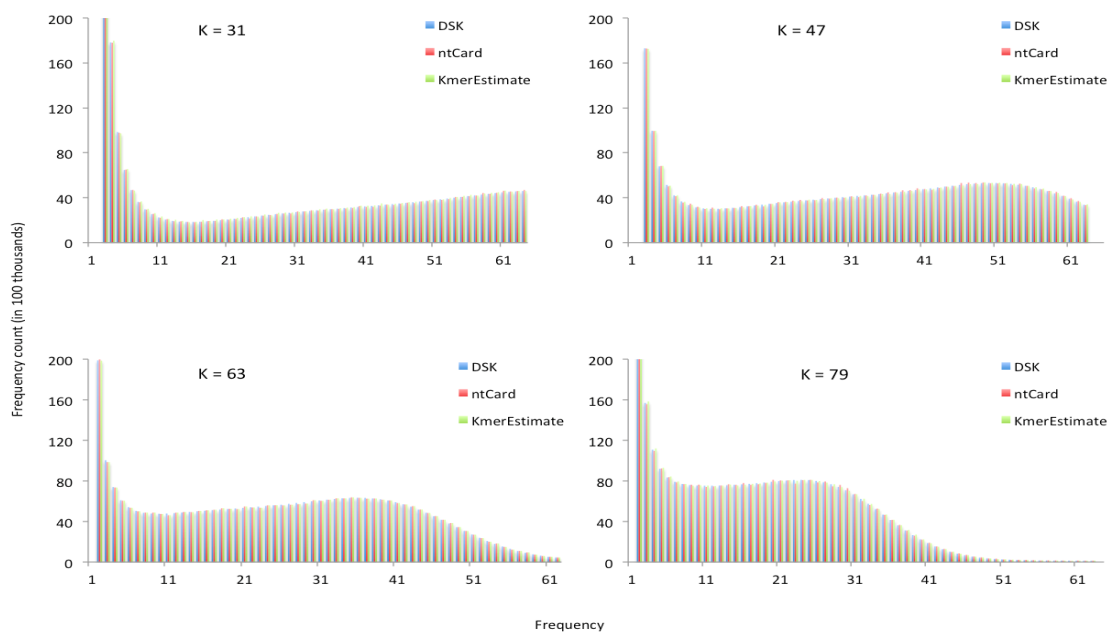


Figure 2.4: k -mer count histogram for *Bombus impatiens* reads

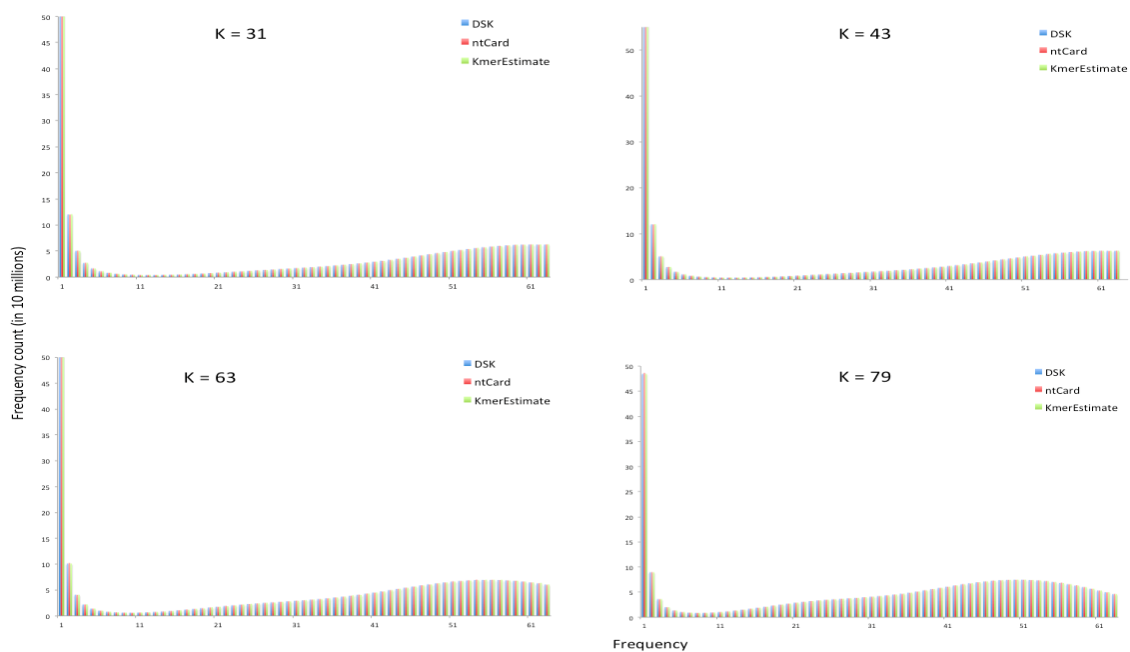
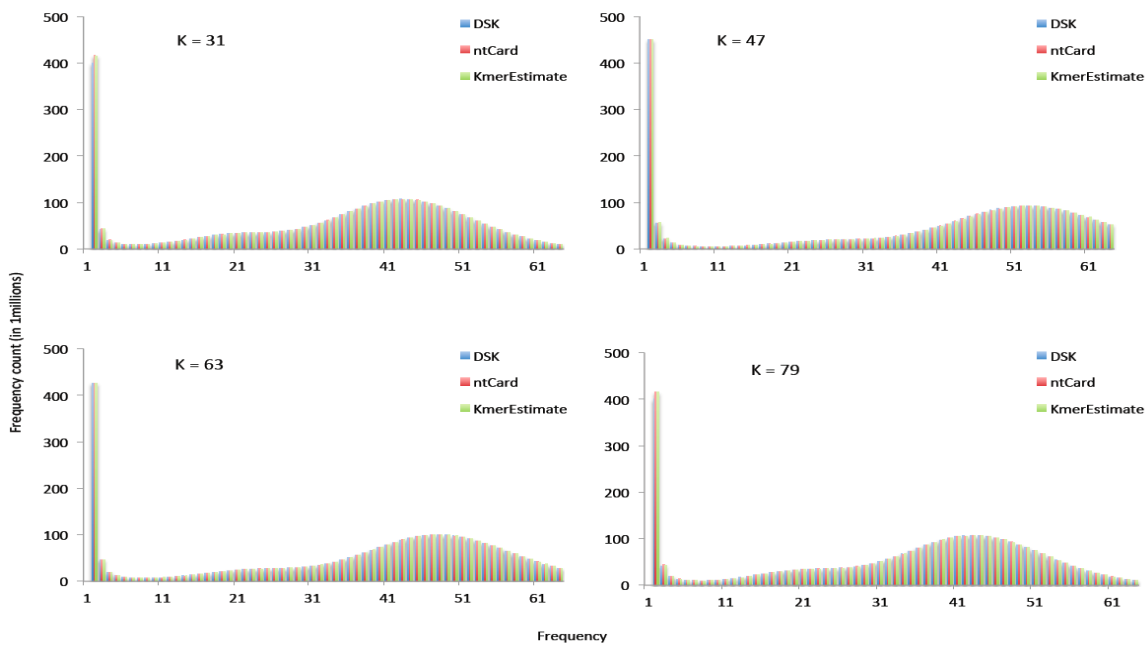
Figure 2.5: k -mer count histogram for NA19238 readsFigure 2.6: k -mer count histogram for HG004 reads

Table 2.1: Dataset specification

Dataset	Total # of reads	Read length	Total bases	Size
HG14	36,504,800	101 bp	3,686,984,800	7.8 GB
Bumblebee	303,118,594	124 bp	37,586,705,656	92 GB
NA19238	456,979,900	500 bp	228,489,950,000	462 GB
HG004	868,593,056	250 bp	217,148,264,000	448 GB

2.3.2 Accuracy

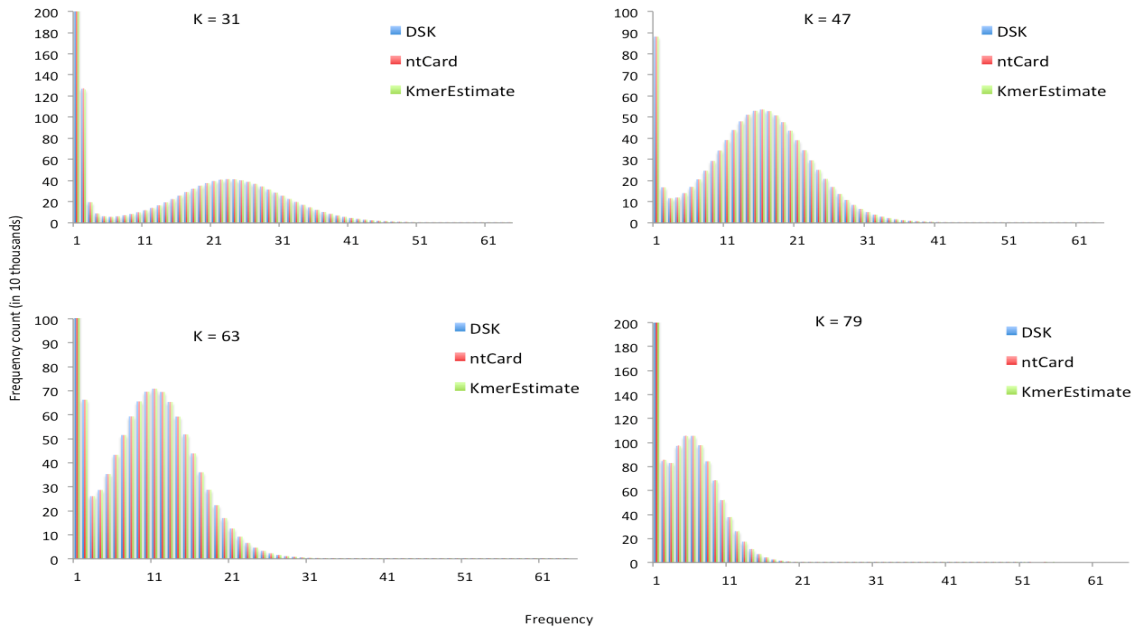
The results for number of singleton k -mers (f_1), distinct k -mers (F_0) and error percentages are shown in tables 2.2-2.5. The error percentages of estimated counts of ntCard, KmerStream and KmerEstimate were calculated based on DSK results.

KmerStream has higher error rates compared to ntCard and KmerEstimate for all four datasets. Compared to ntCard, KmerEstimate has lower error rates in most of the experiments as shown by bold entries in Tables 2.2-2.5. Our algorithm estimates the counts with error rates $\leq 0.6\%$ in all the 16 experiments. (4 datasets, 4 k -mer sizes).

The results of k -mer frequency histograms of DSK, ntCard and KmerEstimate are shown in Fig 2.7-2.10. KmerStream is not included as it does not estimate the number of k -mers with frequency ≥ 2 . The frequency histograms are drawn using f_2 - f_{64} . The results show that the frequency histograms of both ntCard and KmerEstimate are almost accurate as compared to DSK.

Table 2.2: Accuracy of algorithms in estimating F_0 and f_1 for HG14 reads

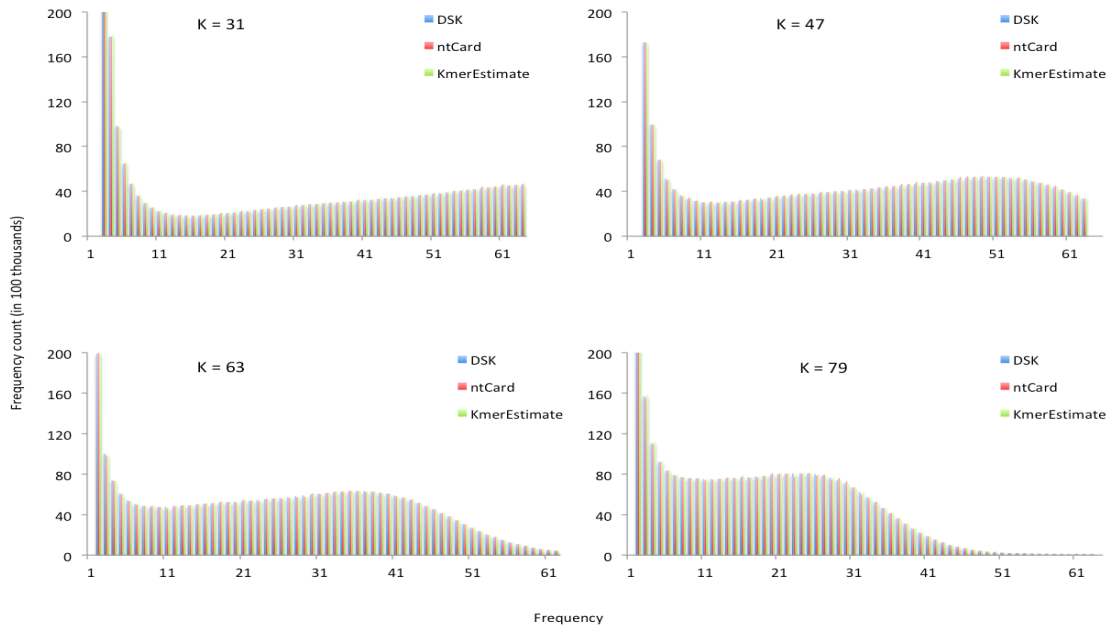
k	F_0/f_1	DSK	ntCard	Error(%)	KmerStream	Error(%)	KmerEstimate	Error(%)
31	f_1	372,088,750	372,502,884	0.1113	358,956,880	3.5292	372,163,712	0.0201
	F_0	472,030,322	472,542,056	0.1084	461,217,054	2.2908	472,005,440	0.0053
47	f_1	385,778,023	386,070,380	0.0758	379,674,696	1.5821	385,892,800	0.0298
	F_0	484,389,437	484,849,139	0.0949	479,027,609	1.1069	484,519,296	0.0268
63	f_1	336,752,336	337,125,461	0.1108	331,846,491	1.4568	336,800,384	0.0143
	F_0	432,569,742	433,072,480	0.1162	429,572,210	0.693	432,633,440	0.0147
79	f_1	240,303,417	240,137,776	0.0689	238,103,508	0.9155	240,009,168	0.1224
	F_0	329,415,228	329,445,681	0.0092	328,564,210	0.2583	329,235,664	0.0545

Figure 2.7: k -mer count histogram for HG14 readsTable 2.3: Accuracy of algorithms in estimating F_0 and f_1 for Bumblebee reads

k	F_0/f_1	DSK	ntCard	Error(%)	KmerStream	Error(%)	KmerEstimate	Error(%)
31	f_1	4,643,105,571	4,648,296,087	0.1118	4417927345	4.8497	4,642,798,080	0.0066
	F_0	5,188,072,759	5,192,697,532	0.0891	4,991,000,112	3.7986	5,187,693,312	0.0073
47	f_1	5,055,109,675	5,581,373,383	0.0762	4,935,411,032	2.3679	5,056,776,192	0.33
	F_0	5,584,417,019	5,051,258,339	0.0545	5,475,881,764	1.9435	5,586,009,600	0.0285
63	f_1	5,002,185,750	5,003,020,481	0.0167	4,925,922,211	1.5246	5,001,567,744	0.0124
	F_0	5,501,837,913	5,502,587,136	0.0136	5,437,010,148	1.1783	5,501,789,184	0.0009
79	f_1	4,563,091,728	4,537,240,138	0.5665	4,518,206,449	0.9837	4,538,720,512	0.5341
	F_0	5,013,470,804	4,989,930,287	0.4695	4,967,653,605	0.9139	4,992,864,000	0.411

2.3.3 Time and space

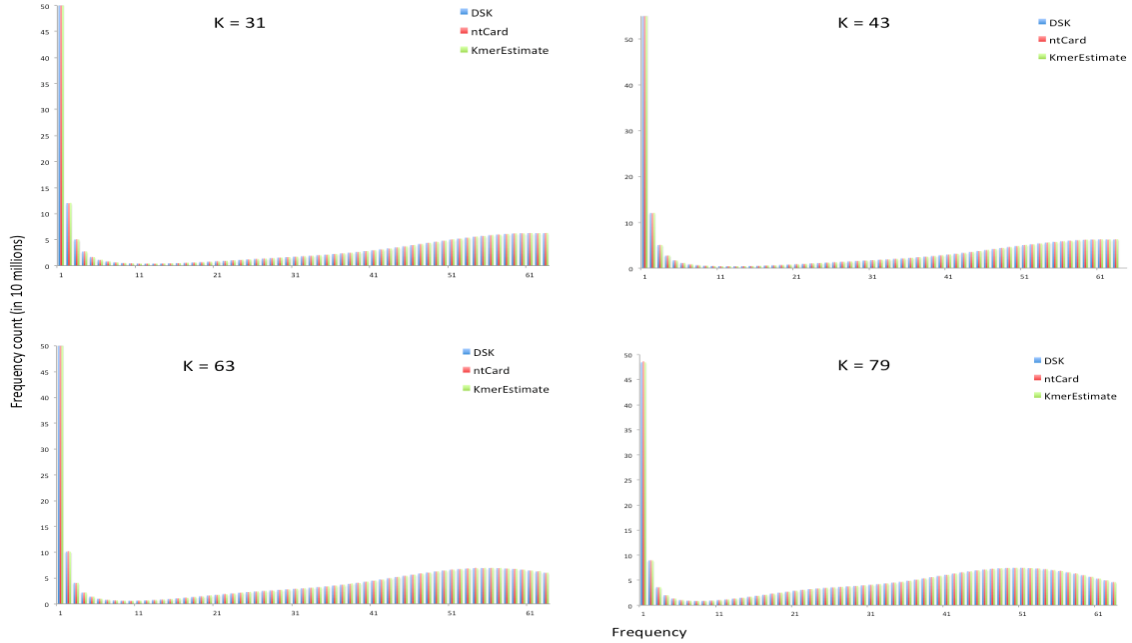
It is shown in [95], ntCard uses 500 MB of RAM for computing the full k -mer frequency histogram. It is significantly memory-efficient as compared to DSK. As DSK counts the frequency of each solid k -mers i.e. k -mers with frequency ≥ 1 , it is expected that more memory would be used. The memory requirement for KmerEstimate depends on given sample size. We observed that a good sample size requires less than 450 MB of RAM that is slightly better than ntCard's memory usage. The current implementation of KmerEstimate uses an array of hashmaps to store the sampled k -

Figure 2.8: k -mer count histogram for Bumblebee readsTable 2.4: Accuracy of algorithms in estimating F_0 and f_1 for NA19238 reads

k	F_0/f_1	DSK	ntCard	Error(%)	KmerStream	Error(%)	KmerEstimate	Error(%)
31	f_1	27,062,279,171	27,065,865,034	0.0133	24,675,257,170	8.8205	27,058,696,192	0.0132
	F_0	30,475,635,517	30,479,602,836	0.013	28,632,428,352	0.0481	30,473,578,496	0.0067
47	f_1	37,672,547,800	37,670,862,086	0.0045	37,215,397,047	1.2135	37,679,218,688	0.0177
	F_0	41,232,028,026	41,228,865,394	0.0077	40,834,786,551	0.9634	41,239,109,632	0.0172
63	f_1	46,452,268,585	46,442,644,573	0.0207	46,183,895,819	0.5777	46,450,513,920	0.0038
	F_0	50,089,031,424	50,080,598,102	0.0168	49,979,319,5526	0.5906	50,088,712,192	0.0006
79	f_1	54,321,071,396	54,241,229,869	0.147	54,132,773,870	0.3466	54,263,234,560	0.1065
	F_0	57,995,538,410	57,927,158,209	0.1179	57,838,450,523	0.2709	57,948,172,288	0.0817

mers whereas ntCard uses an array. Using space-efficient hashmaps, we could further improve memory usage of KmerEstimate.

The runtime of KmerEstimate is almost similar to ntCard. It takes about 40 minutes to estimate k -mer frequency histograms of both the human dataset (HG004, NA19238). The current implementation of KmerEstimate does not support multi-threading, so we run all our experiments on a single core. The ntCard algorithm takes about 6 minutes to compute k -mer frequency histogram for human genome dataset when 12 cores are used. But it took around 46 minutes when it was run on a

Figure 2.9: k -mer count histogram for NA19238 readsTable 2.5: Accuracy of algorithms in estimating F_0 and f_1 for HG004 reads

k	F_0/f_1	DSK	ntCard	Error(%)	KmerStream	Error(%)	KmerEstimate	Error(%)
31	f_1	13,040,267,779	13,040,041,802	0.0017	11,743,039,453	9.9479	13,042,160,640	0.0145
	F_0	16,245,764,745	16,245,869,976	0.0006	15,102,059,608	7.04	16,246,677,504	0.0056
47	f_1	16,254,677,761	16,258,434,303	0.0231	15,862,419,428	2.4132	16,253,229,056	0.0089
	F_0	19,619,791,781	19,624,783,331	0.0254	19,237,376,370	1.9491	19,620,672,512	0.0045
63	f_1	17,832,146,715	17,836,597,009	0.025	17,605,947,255	1.2685	17,833,374,720	0.0069
	F_0	21,273,945,928	21,276,906,756	0.0139	21,085,818,688	0.8843	21,276,222,464	0.0107
79	f_1	18,592,930,567	18,526,819,210	0.3556	18,435,991,949	0.8441	18,522,447,872	0.3791
	F_0	22,070,254,735	22,017,619,287	0.2385	21,939,564,081	0.5922	21,276,222,464	0.0134

single core. Both ntCard and KmerEstimate is almost 15x faster than KmerStream that computes only F_0 and f_1 .

2.3.4 Sample size

The ntCard uses r trailing bits of 64 bit hash value to sample the k -mers. In their implementation, the r value is fixed to 27 that means the maximum sample size is $2^{27} \approx 135$ millions. By observing the memory usage vs sample size, we found that a sample of 25 millions approximates the k -mer frequency counts with $\leq 0.6\%$ that is

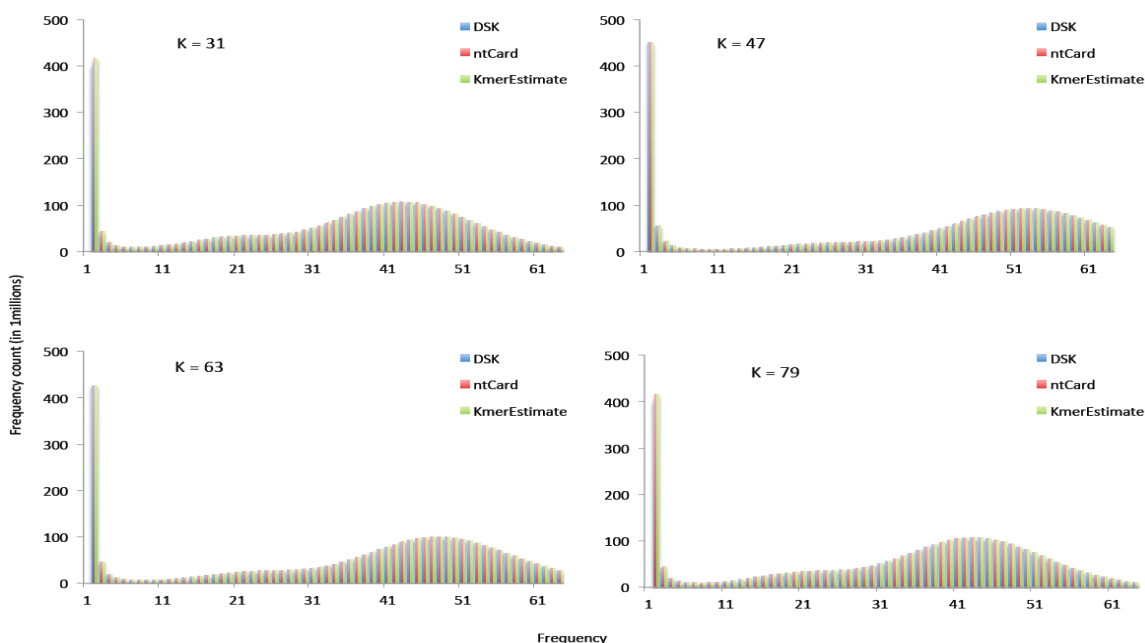


Figure 2.10: k -mer count histogram for HG004 reads

comparable to ntCard results. So, KmerEstimate performed better than ntCard by using 7x less sampled k -mers.

2.4 Conclusion

With the availability of low cost sequencing technologies, more and more data is being produced for studying different organisms. However, the handling of large amount of data needs efficient computational approaches. The streaming algorithms has been proven to be both space-efficient and time-efficient for computing approximate values of a function while working with large-scale dataset. It uses a very small amount of memory and is significantly faster as compared to algorithms that computes exact results.

We developed a streaming algorithm, KmerEstimate, that approximates k -mer abundance histogram. The k -mer abundance histogram is very useful in genome and

transcriptome assembly, error correction of sequencing reads, and sequence alignments. In this algorithm, we employed the techniques used in the BJKST algorithm and approximates the abundance histogram by sampling a small number of k -mers from a large stream of k -mers. The results of our algorithm is within 0.6% error rate that is better than other streaming approaches used so far for this problem. It uses less memory than ntCard as the size of the sample is $(1/7)^{th}$ of the sample size used in later algorithm. Moreover, we also prove theoretical guarantees on our algorithm. We expect that KmerEstimate can be used for estimating genome characteristics, error correction, copy number variation and various k -mer based downstream analysis. Our future work includes improving the memory usage by using better space efficient hashmaps for storing the sampled k -mers and updating our tool that will support multi-threading.

Chapter 3

Discovery of conserved non-coding sequences efficiently

Publications:

- Xianjun Lai[†], Sairam Behera[†], Zhikai Liang, Yanli Lu , Jitender S. Deogun, James C. Schnable, 2017, “STAG-CNS: An Order-Aware Conserved Noncoding Sequences Discovery Tool for Arbitrary Numbers of Species”, *Molecular Plant*,10(7), pp. 990-999. doi: 10.1016/j.molp.2017.05.010.

[†] Joint first authors.

Author note: I developed and implemented the algorithm and helped X. lai in running the program and analysis.

- Sairam Behera, Xianjun Lai, James C. Schnable and Jitender S. Deogun. 2018. “DiCE: Discovery of conserved noncoding sequences efficiently”, *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Kansas City, MO, USA, pp. 79-82, doi: 10.1109/BIBM.2017.8217628.

3.1 Introduction

The information in the functional DNAs are used for the transcription of RNAs that leads to synthesis of proteins. This process is called gene expression that produces the functional gene products. In related species, the sequences that regulate the

gene expression tend to be similar as compared to non-functional DNAs. These sequences are called conserved non-coding sequences (CNS) and are highly associated with transcription factor binding sites and cis-acting regulatory elements. The CNSs like other small genomic regions lack well-defined signatures. Therefore, it has been a challenging problem in both functional and evolutionary genomics to determine those parts of genome that are under selective constraint [59]. It may be noted that the genes have well-defined signatures. In animals, many CNSs are large (100bp)[130], while different analyses in plants have primarily identified smaller 15–50 bp CNSs. Therefore, the identification of CNS in plants is more challenging compared to animals [133] [10].

Comparative genomics is the field of biological research in which the genomes of similar species are compared to infer the biological functions. By using many computational approaches on the analysis of genomic features that are conserved in multiple organisms over million of years, researchers are able to pinpoint the signals that control gene functions. The comparative genomics exploits the similarities as well as differences in the proteins, RNA, and regulatory regions of different organisms to infer how selection has acted upon these elements.

The regulation of gene expression in plants has been studied using comparative genomics and gene expression data [100]. The conserved non-coding sequences are the regions close to the genes which do not take part in the transcription process i.e. do not code for proteins. However, it has been observed that these regions are functionally constrained and show high levels of sequence conservation between orthologs of multiple species. It has been discovered that these sequences are involved in regulation of gene expression. The functions of specific conserved noncoding sequences still remain unknown for many cases. Therefore, it is desired to identify the conserved noncoding regions to study their functions and mechanisms of gene-regulation.

Most of the existing approaches for the discovery of CNSs are based on either multiple sequence alignment or multiple pairwise alignment. These approaches are not efficient when large number of species are used. Moreover, finding small CNSs ($\leq 15\text{bp}$) present in plants with high accuracy is still a bottleneck for these approaches. The multiple sequence alignment algorithms are based on the assumption that most of the sequences are homologous. This makes a perfect sense for the protein sequences where insertions or deletions are rare. However, the majority of sequences in plant noncoding regions are non-homologous and have been shuffled over evolutionary time scales by insertion and deletion of transposons. The objective of CNS discovery problem is to find small islands of conserved sequences within that sea of non-conserved sequences.

In this chapter, we present a novel algorithm for identifying CNSs in closely related species especially the plant species where CNSs are very small. This algorithm is based on the suffix tree and maximum weighted path in a directed acyclic graph (DAG). It exploits the relationships between the conserved noncoding sequences (CNSs) and the maximal exact matches (MEMs) that can be derived from suffix tree of sequences. In our algorithm, the MEMs of sequences and the ordering of their locations are represented in a weighted directed acyclic graph. The polynomial time algorithm for finding maximum weighted path(s) in a DAG is used to determine a set of CNSs from the MEMs. The use of suffix tree in this algorithm makes it independent of pairwise alignments, thus avoids quadratic number of sequence alignments. We present two algorithms, first for discovering exactly matched CNSs and second for CNSs with a given rate of mismatches. The exact matched algorithm was used for comparative genomic analysis of multiple grass species.

We tested our algorithm to find CNSs in the promoter regions of 17,996 syntenic genes of six grass species [74]. The results demonstrate that as larger number of

species is used in the comparison, CNSs with smaller sizes can be identified with high level of confidence as well as high level of false positive tolerance. The comparison of our algorithm with the best known pairwise alignment based CNS discovery pipeline (CDP) [138] shows that our approach discovers almost 500 more CNSs that are not identified by other approach. Also, the runtime of our algorithm is 5 times faster than CDP. The accuracy of the CNSs are validated using permutation tests and comparing with the available DNase Hypersensitivity Sites (DHS) data of rice callus and rice seeding tissues.

3.2 Background and Related Works

In this section, we briefly explain the data structures and notations used in our algorithm. Also, we discuss some of the existing methods that have been used for the discovery of CNSs.

A sequence S of length m is a string of m characters of a given alphabet set Σ . A subsequence, denoted as $S[i, j]$ where $1 \leq i \leq j \leq m$, is a collection of contiguous characters from i^{th} position to j^{th} position in S . Therefore, the sequence S can also be denoted as $S[1, m]$. The suffix of a sequence is a subsequence that ends at the last position of the sequence. Given a sequence $S[1, m]$, each subsequence $S[i, m]$, where $1 \leq i \leq m$, is called a suffix of sequence S . Similarly, a prefix is defined to be a subsequence starting at the first position i.e. $S[1, i]$, where $1 \leq i \leq m$.

For biological sequences, $\Sigma = \{A, C, G, T\}$ i.e. the alphabet set contains four characters A , C , G and T . Following are few definitions of terms used in our algorithm.

Definition 1: (Maximal Exact Match) Given a sequence $S[1, m]$, two subse-

quences $S[i_1, j_1]$ and $S[i_2, j_2]$ with $i_1 \neq i_2$ are called repeats if $S[i_1, j_1] = S[i_2, j_2]$. A pair of repeats are left-maximal or right-maximal if $S[i_1 - 1] \neq S[i_2 - 1]$ or $S[j_1 + 1] \neq S[j_2 + 1]$, respectively. The maximal exact matches (MEMs) are the repeats (within a sequence or among several sequences) that are left- and right-maximal.

Definition 2: (Suffix Tree) Given a sequence $S[1, m]$, the suffix tree for S , denoted as $ST(S)$ or ST , is a trie data structure that stores all suffixes of S in a compressed manner. It is a rooted tree with m leaves and each leaf is labeled with an index i that represents the suffix starting at i^{th} position i.e. $S[i, m]$. A suffix link is a link between two internal nodes $N_1 \rightarrow N_2$ with N_1 represents string aS and N_2 represents string S , where a is a single character and S is a possibly empty string. Suffix Tree was first studied by Gusfield [54] and has been used in variety of applications in computational biology including: search applications, single sequence analysis applications, and multiple sequence analysis applications [19].

Definition 3: (Generalized Suffix Tree) Given a set of n sequences $\mathbb{S} = \{S_1, S_2, \dots, S_n\}$, a generalized suffix tree of a set of sequences \mathbb{S} , denoted $GST(\mathbb{S})$ or simply GST , is a trie data structure that stores all suffixes of all the sequences in a compressed manner. The leaf nodes are labeled by an integer pair (i, j) denoting suffix starting from position j in i^{th} sequence S_i i.e. $S_i[j, m_i]$ where m_i is the length of sequence S_i .

Given a set of n sequences. For an integer k , $\mathbb{M}_{\geq k, l}$ denotes a maximal exact match (MEM) with length $\geq k$ that is present in at least l different sequences where $1 < l \leq n$. Thus, $\mathbb{M}_{\geq k, n}$ is a MEM of length at least k that is present in all n sequences. Two MEMs $\mathbb{M}_{\geq k, n}^1$ and $\mathbb{M}_{\geq k, n}^2$ are called non-intersecting MEMs if the end positions of all repeat fragments of one MEM is strictly less than the start positions of repeat

fragments of other MEM. Similarly, two MEMs are called overlapping MEMs if at least one of the repeat fragments of a MEM starts at a position that lies between start and end positions of other MEM. Two non-overlapping and non-intersecting MEMs are called independent MEMs. An independent set of MEMs is a collection of MEMs where no two MEMs overlap or intersect each other. Figure 3.1 illustrates the concept of overlapping, intersecting and independent MEMs. The MEM 1 (green color) intersects the MEM 2 (red color). Similarly, MEM 3 intersects both MEM 4 and MEM 5. The MEM 4 and MEM 5 are the example of overlapping MEMs. A set consisting of MEM 1, MEM 3 and MEM 6 is an example of independent set of MEMs.

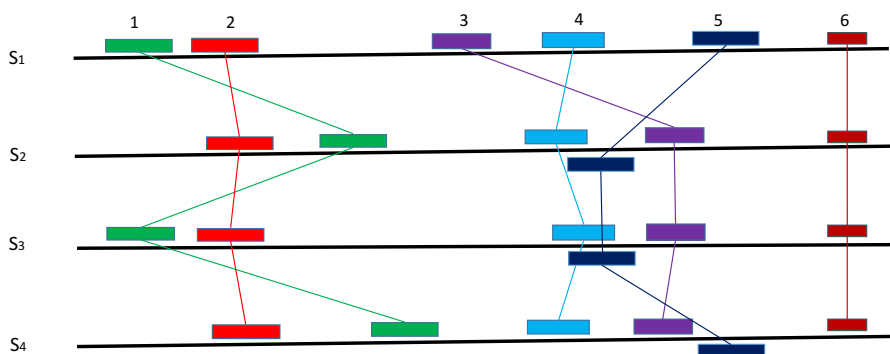


Figure 3.1: Intersecting, overlapping, and independent MEMs

A suffix tree (or generalized suffix tree) can be constructed in linear time and space using suffix links [139]. The internal nodes of a suffix tree that are lowest common ancestors of at least l leaf nodes belonging to l different sequences represent MEMs. The internal nodes representing MEM nodes are called MEM nodes.

Several approaches are used to identify the small CNSs in plants. Most of the approaches are based on either multiple pairwise alignments or multiple sequence alignments. Thomas et al. worked on *Arabidopsis thaliana* plant species to discover CNSs with length ranging from 15 to 285 bp by manual inspection of syntenic gene

pairs [133]. The pairwise genome alignments using aligners like BLAST[4], QUOTA-ALIGN[131], LASTZ[57] etc. are also used for identifying CNSs [138] [46]. Baxter et al. used a seaweed algorithm [134] based fast implementation of alignment plot method to find CNSs by global alignment of orthologous promoter regions [10]. The whole genome alignment methods using progressive alignment programs are also used to discover the CNSs in plant species [59]. The detection of CNSs using a comparative motif mapping and alignment-based phylogentic footprinting is used for dicot plant genomes [140]. Turco et al. designed a pipeline based on pairwise comparisons for detecting the CNSs in closely related grass genomes[138]. The pairwise BLAST search result among all pairs are used to identify the regions with significance greater than a 15bp exact match.

3.3 Methodology

In this section, we describe our algorithm to identify the CNSs from a set of given sequences of closely related species.

The transcribed regions i.e. coding sequences of a DNA that take part in the transcription process exhibit slower rate of mutations during the evolution process. On the other-hand, it has been well-established that the noncoding sequences have higher rate of mutations. A conserved noncoding sequences (CNSs) is a DNA sequence that is conserved across non-transcribed regions of a large number of species. Many researches have discovered that the CNSs are present in the both adjacent regions i.e. upstream and downstream of a coding sequence. The identification of CNSs requires comparison of long genomic DNA sequences of related species. Given the genome sequences of two or more species and the annotation file that contains the

information about coding regions, the problem of CNS identification is to find all small order-consistent conserved regions that exist in the upstream and downstream regions of homologous genes.

3.3.1 Problem definition

A set of conserved non-coding sequences (CNSs) is a set of independent MEMs that are present in non-genomic regions of all n sequences. If \mathcal{M} is a set of all MEMs that are present in all n sequence i.e. $\mathcal{M} = \{\text{all } \mathbb{M}_{\geq k, n}\}$ and \mathcal{C} is the set of CNSs, then $\mathcal{C} \subseteq \mathcal{M}$. The score of \mathcal{C} , denoted as $score(\mathcal{C})$, is defined as the total length of all CNSs present in that set. Let $\mathcal{C} = \{C_1, C_2 \dots C_m\}$, where $C_i \in \mathcal{M}$, be a set of m CNSs and $|C_i|$ denotes the length of sequence associated with C_i , then $score(\mathcal{C})$ is defined below

$$score(\mathcal{C}) = \sum_{i=1}^m |C_i|$$

Following is the formal definition of the CNS identification problem.

Given $\mathbb{S} = \{S_1, S_2 \dots S_n\}$ a set of n sequences, find a set \mathcal{C} that has maximum *score*.

3.3.2 Algorithm

The main idea of our algorithm is identifying a set of CNSs from the set of MEM nodes in the suffix tree of sequences. Given a target minimum CNS length k , the objective is to extract the set \mathcal{M} i.e. set of MEM $\mathbb{M}_{\geq k, n}$. The MEMs in \mathcal{M} are used to construct a weighted directed acyclic graph (DAG). The set of MEMs associated with the maximum weighted path in the DAG defines a set \mathcal{C} of CNSs that has maximum

score.

Algorithm 1: Finding CNS

Input: Set of n sequences $S = \{S_1, S_2, \dots, S_n\}$

Output: A set of CNSs

- 1 Construct Generalized Suffix Tree \mathcal{T} of sequences $S_1, S_2 \dots S_n$ with suffix links
 - 2 Mark the MEM nodes that represent $\mathbb{M}_{\geq k, n}$ and are not present in genomic regions
 - 3 Extract the MEMs from the marked nodes in \mathcal{T}
 - 4 Construct a weighted directed acyclic graph (DAG) using MEMs
 - 5 Perform a topological ordering of the weighted DAG
 - 6 Find the maximum weighted path \mathbb{P} in the DAG
 - 7 Store the MEMs associated with \mathbb{P} in set \mathcal{C}
 - 8 Output \mathcal{C}
-

Figure 3.2: Algorithm for identifying CNSs

The pseudo-code of the algorithm is given in Figure 3.2. Our algorithm first constructs a generalized suffix tree (*GST*) for given n sequences using Ukkonen's linear time algorithm [139]. The MEM nodes are marked in the *GST* using the techniques of splitMEM [91]. However, not all MEMs present in *GST* are marked. We are only interested in MEMs of the type $\mathbb{M}_{\geq k, n}$ present in non-genomic regions and the repeat fragments are not in the both sides of genomic region. Then the marked nodes are extracted by traversing the *GST*. Each MEM is given a score that is equal to the length of the subsequence associated with it.

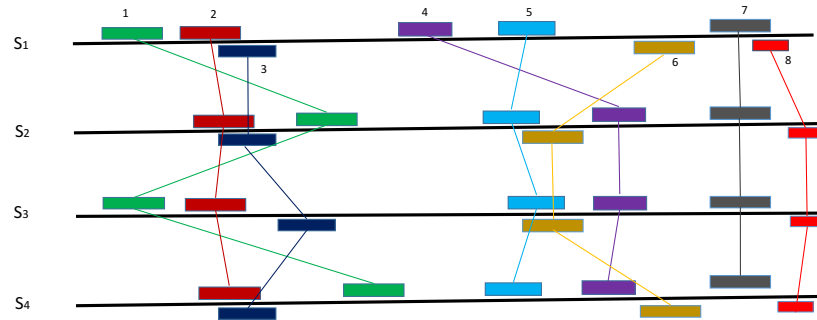
A weighted directed acyclic graph (DAG) is constructed using the MEMs extracted from *GST*. Each node of DAG represents a MEM and the weight of a node is the length of corresponding MEM. For two non-overlapping and non-intersecting MEMs, a directed edge is constructed from the MEM that starts earlier in the sequences to

the other MEM. The transitive edges are deleted. The weight of each directed edge is equal to the weight of the node it is pointing to. Figure 3.3 illustrates an example of MEMs in four different sequences and the corresponding weighted directed acyclic graph. Figure 3.3a shows the positions of MEMs in four sequences, the table in Figure 3.3b contains MEMs, their length, start positions in each sequence and the corresponding node in DAG. The Figure 3.3c is the weighted directed acyclic graph constructed using the informations given in the table in Figure 3.3b.









A source is a node with zero in-degree and a sink is a node with zero out-degree. A super-source node is added to DAG. A directed edge from super-node to each of the source nodes is constructed and the weight of the edge is equal to the weight of source node it is pointing to. Similarly, a super-sink node is also added to DAG. A directed edge from each of the sink nodes to the super-sink node is constructed. The weight of the edge is assigned to 0 (see Figure 3.3c). The algorithm finds maximum weighted path(s) between super-source node (S) and super-sink node (T). The set of MEMs corresponding to the nodes in the maximum weighted path is a set of CNSs, denoted by \mathcal{C} . The ties are broken based on the distance between consecutive CNSs in \mathcal{C} . The Figure 3.4 shows an example of ranking of the set of CNSs if more than one sets of CNSs have same maximum score. Let d_1 , d_2 and d_3 be the distances between two consecutive CNSs. The CNS set is ranked higher if $|d_1 - d_2| + |d_2 - d_3| + |d_3 - d_1|$ is minimum. The highest ranked CNS set is chosen among set of all CNSs with equal score.

3.3.3 CNS with mismatches

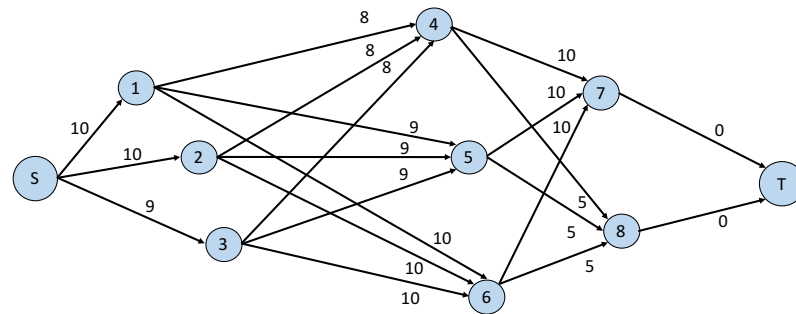
The algorithm described in the previous section identifies a set of CNSs where each CNS is an exactly matched repeat fragment present in all sequences. It is also pos-



(a) MEMs in three sequences

MEM	Length	S ₁ (Start Position)	S ₂ (Start Position)	S ₃ (Start Position)	S ₄ (Start Position)	ID
	10	100	131	100	132	1
	10	115	117	115	117	2
	9	120	120	127	120	3
	8	146	176	176	173	4
	9	160	158	161	159	5
	10	181	165	168	181	6
	10	183	183	183	183	7
	5	190	196	196	195	8

(b) MEM positions and corresponding nodes



(c) DAG

Figure 3.3: MEMs and weighted directed acyclic graph

sible to have conserved noncoding regions with few mismatches. In this section, we present an algorithm for finding all CNSs with a given maximum mismatch rate p . The Algorithm 2 is obtained by modifying the Algorithm 1 and presented in Figure 3.5.

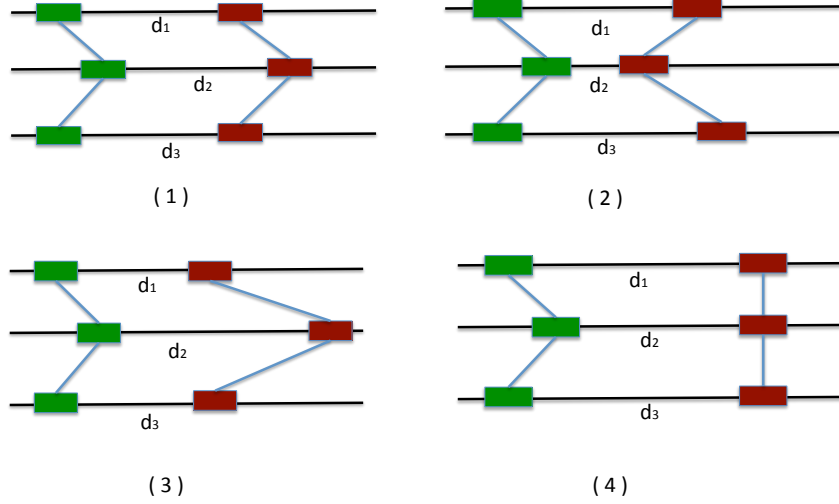


Figure 3.4: Ranking of CNSs

Given two sequences S and T of equal length ℓ and a mismatch rate of ρ , S and T are said to be a match with mismatch rate ρ if number of mismatches in S and T $\leq (\rho/\ell) \times 100$.

Definition 4: ($p\text{MEM}_{\geq k}$) Given an integer p , we define $p\text{MEM}_{\geq k}$ as the maximal repeat fragment of length at least k present in all sequences with pairwise mismatch rate $\rho \leq p$. For $p = 0$, $p\text{MEM}$ becomes an (exact) MEM i.e. $\text{MEM}_{\geq k,n}$.

Following is a brief description about the modifications done in Algorithm 1 to obtain the CNSs with mismatches. In Algorithm 2, we construct GST of sequences and extract both \mathcal{M} i.e. set of all $\text{MEM}_{\geq k,n}$ and set of all $\text{MEM}_{\geq k,l}$, $2 \leq l < n$, denoted by \mathcal{M}' . The $p\text{MEMs}$ are constructed from MEMs in \mathcal{M} and \mathcal{M}' . The algorithm uses the similar DAG based approach to identify a CNS set that includes the CNSs with a mismatch rate $\leq p$. The maximum weighted path in the DAG is a set of CNSs that has maximum *score*.

Algorithm 2: Finding CNS with mismatches

Input: Set of n sequences $S = \{S_1, S_2, \dots, S_n\}$

Output: A set of CNSs

- 1 Construct Generalized Suffix Tree \mathcal{T} of sequences $S_1, S_2 \dots S_n$ with suffix links
 - 2 Mark the MEM nodes that represent $\mathbb{M}_{\geq k, l}$, where $2 \leq l \leq n$, and are not present in genomic regions
 - 3 Extract the MEMs from the marked nodes in \mathcal{T}
 - 4 Check if MEMs of type $\mathbb{M}_{\geq k, n}$ can be extended to construct p MEMs
 - 5 Construct a weighted directed acyclic graph (DAG) using MEMs and p MEMs
 - 6 Perform a topological ordering of the weighted DAG
 - 7 Find the maximum weighted path \mathbb{P} in the DAG
 - 8 Store the MEMs and p MEMs associated with \mathbb{P} in set \mathcal{C}
 - 9 Output \mathcal{C}
-

Figure 3.5: Algorithm for finding exact-matched and mismatched CNSs

3.4 Experimental Results

We implemented Algorithm 1 and 2 in C++ and it is available at <https://github.com/srbehera11/DiCE> as the DiCE software. An earlier version of the algorithm, called STAG-CNS, that can identify order-aware CNSs is available at <https://github.com/srbehera11/STAG-CNS>. The algorithm requires three main parameters: the fasta file containing gene sequences and their locations in the chromosome, the minimum CNS length (k) and threshold mismatch rate (p). The output is a file containing a set of CNSs with their lengths and starting positions in each sequences. The algorithm also output three other files that are used in three different visualization softwares. The algorithm has been tested on a single core of a Xeon E5-2697 v4 2.3GHz server with 2 CPU/36 cores per node and a total of 512GB of RAM at

Holland Computing Center (HCC), University of Nebraska-Lincoln.

In the following sections, we discuss the various results obtained from the analysis of six closely related grass species. In Section A, we discuss the accuracy and sensitivity of the CNSs identified using our approach. The comparative analysis of three grass species using our approach and CDP is given in Section B. In Section C, the identified CNSs are validated using available DNase Hypersensitive sites of two different tissues of rice. A comparative analysis of the run time performances of DiCE and CDP is given in section D.

3.4.1 Accuracy and sensitivity of our approach

Our algorithm is tested on a set of six grass species that are completely sequenced i.e. the genomes and annotation files are available. We selected a set of 200 genes, conserved at syntenic orthologous in sorghum, rice, setaria, brachypodium, oropetium, and dichanthelium, from a previously published syntenic gene list [121]. First, the exact matched CNSs (with $p = 0$) are identified among syntenic orthologous genes in three species – sorghum, rice, and setaria – using k i.e. minimum CNS length between 8 and 22 bp. The average false positive discovery rates per gene are estimated using 100 random permutations of the dataset. The result of permutation test shows that the false positives decreases and true positives increases with higher k values (Figure 3.6). The percentage of true positives reaches 99% when the $k = 12$ that explains the accuracy of CNSs.

It is expected that the lengths of CNSs becomes shorter when the number of species is increased. We tested our algorithm on the same set of 200 genes in 2, 3, 4, 5 and 6 species respectively to determine the minimum CNS lengths that gives $\geq 95\%$ true positive discovery rate. For two species, $\geq 95\%$ true positive discovery rate is achieved by $k = 22$ and for six species, it is achieved for $k = 9$ (Figure 3.7).

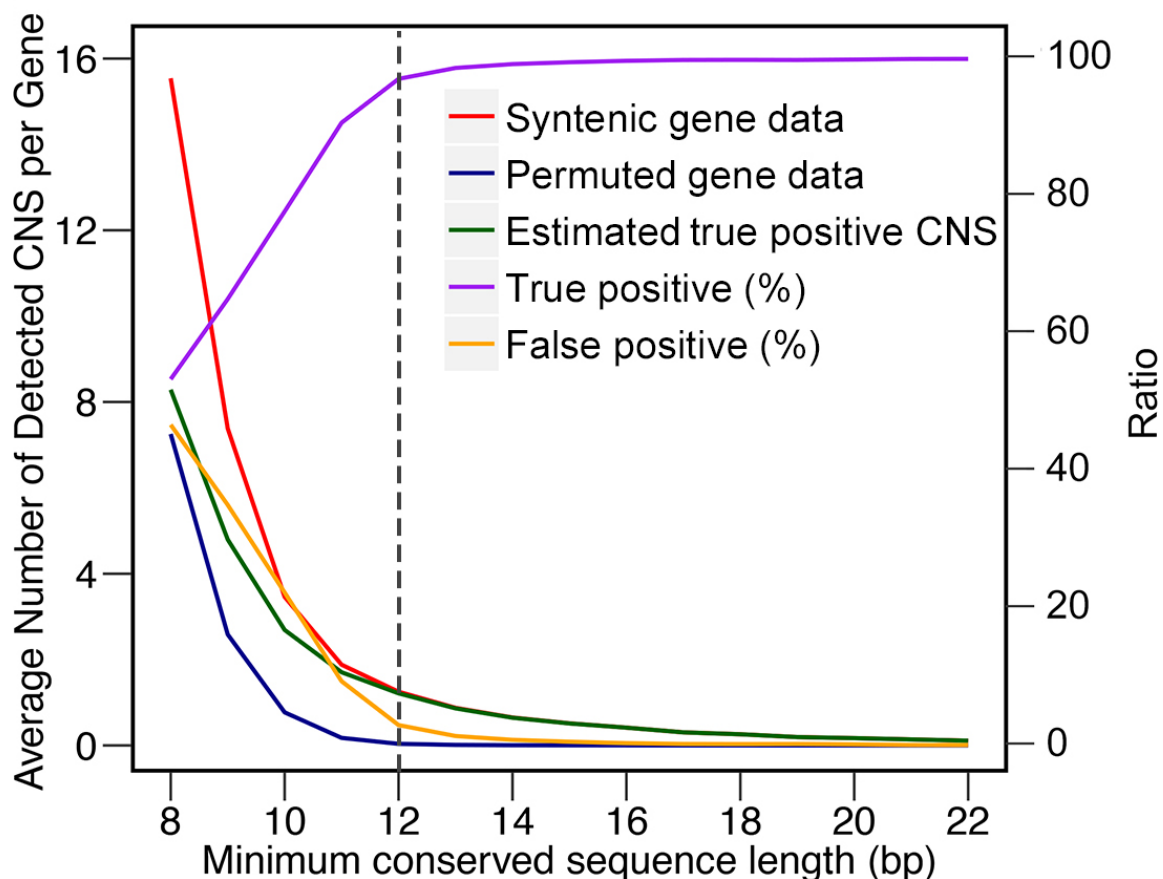


Figure 3.6: True positive discovery rate

3.4.2 Comparison of results from our approach and CDP

The CNS Discovery Pipeline (CDP) is one of the tools used by many researchers in the field of comparative genomics to identify CNSs among closely related species [138]. Unlike Our approach, the CDP performs pairwise comparisons based on BLASTN, and then identifies CNS present in three or more species through overlap with a single common reference.

We selected a set of 17,996 orthologous syntenic genes in sorghum, setaria, and rice that are previously shown to be CNS rich. The CNSs are identified using both our method and CDP (sorghum and rice, sorghum and setaria, and then the pan-grass

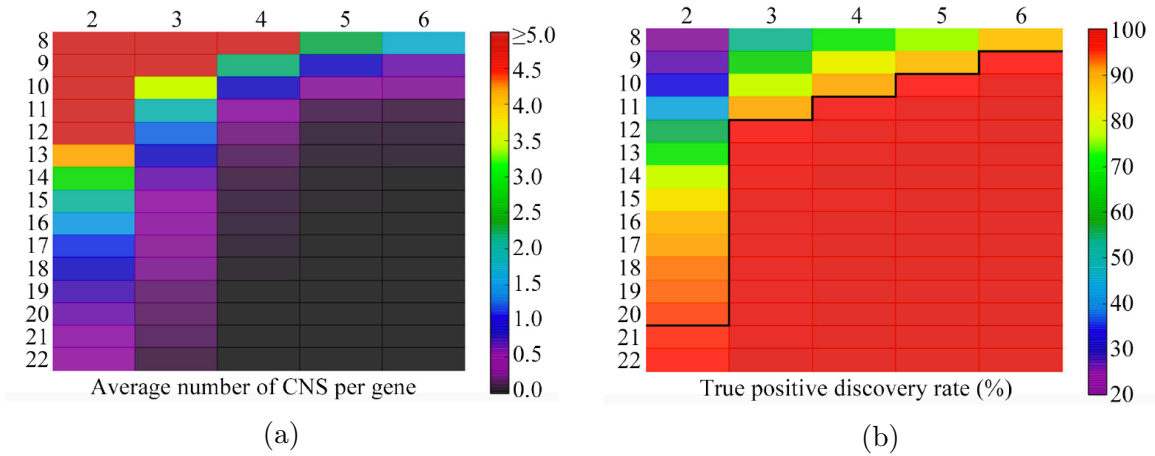


Figure 3.7: (a) The number of CNSs identified in different number of species with different minimum CNS length (b) True positive discovery rate of CNSs across different number of species

Table 3.1: Summary of CNS distribution in 17,996 Orthologous syntenic genes in rice, soybean and sorghum

	CDP	DiCE (p=0)	DiCE (p=15)
Total # Orthologous CNSs	22,139	10,489	22,590
# syntenic genes (at least one CNS)	7,428	4,968	8,142
	(41.27%)	(27.91%)	(45.24%)
Average # CNSs (per gene)	1.23	0.58	1.25
Mean length of CNSs (in bp)	35.76	32.66	44.72
Median length of CNSs (in bp)	27	18	32
Total length of CNS (in bp)	791,640	162,250	804,817

species). The CDP was run with default parameters and our method was run with $k = 6$ and $p = 15$. The CNS information from both methods is summarized in Table 3.1.

The mismatch rate (ρ) for CNSs identified by the CDP is 9.3% for sorghum–rice and 11.6% for sorghum–setaria. The average mismatch rate of CDP is almost 10%. As expected, DiCE identified fewer CNSs in sorghum–rice–setaria than CDP when it is tested with $p = 0$ i.e. no mismatches allowed. However, more number of CNSs are identified DiCE when it is tested with $p = 15$ i.e. mismatch rate $\rho \leq p$ and minimum

CNS length $k = 6$. To maintain the average mismatch rate of 10, DiCE is tested with p value ranging from 10 to 20. The Table 3.1 shows a comparison of results for both DiCE and CDP.

3.4.3 Association of CNSs with DNase hypersensitive sites

It is known that, the regions of open or accessible chromatin zones are functionally related to transcriptional activities. Thus, the regulatory sequences are related to these chromatin regions[137][116]. The open chromatins can be assayed in a whole genome fashion using a range of techniques including FAIRE-seq, MNase-seq, and DNase1 hypersensitivity-seq [155][116]. To validate the accuracy of the CNSs identified by DiCE, the overlap between CNSs and open chromatin regions was tested using a pre-existing set of DNase hypersensitive sites (DH sites) generated from rice seedling and callus tissue [155]. A total of 8,934 CNSs identified from the syntenic genes in sorghum, rice, and setaria with the minimum CNS length 12 bp are compared with the DH sites and the overlap information is shown in Figure 3.8.

It is found that 34.0% (3,037) and 58.1% (5,190) of CNSs overlapped with DH sites identified in seedling and callus tissues respectively. The overlap between CNSs and DHSs that are not found within the 1,000 bp of annotated transcription start-sites shows that 1,130 and 2,289 CNSs overlapped with DH sites identified in seedling and callus tissues respectively. These overlaps are significantly higher than the overlap of CNSs identified by the CDP between rice and sorghum and the same rice open chromatin datasets (25.7% and 41.6% for seedling and callus tissue respectively) [155].

A set of 1,873 rice genes with conserved syntenic orthologs across all 6 similar grass species were used to test how the relative overlap between CNSs identified by DiCE and DH open chromatin regions responded to variation in the number of species and

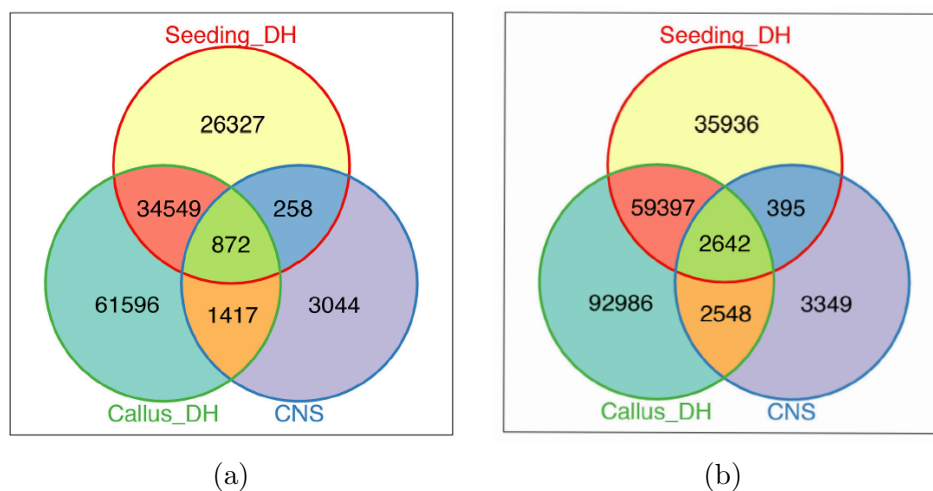


Figure 3.8: (a) Overlap of CNSs of rice, sorghum and setaria with DNase1 hypersensitivity sites (DHS) in rice seedings and rice callus (b) Excludes CNSs & DHSs within 1kb of transcription start sites

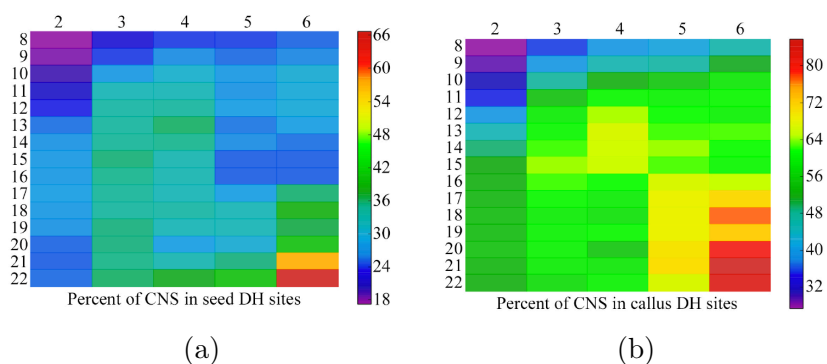


Figure 3.9: Overlap rate of CNSs and DHSs in rice seedings and callus

minimum CNS length. The overlap between potential regulatory sites identified using the DiCE and potential regulatory sites identified using DNase1 hypersensitivity-seq increases either when the number of species used in the DiCE analysis is increased or the minimum length of the CNS is increased (Figure 3.9).

3.4.4 Running time

Both CDP and DiCE were run on a single core of Xeon E5-2697 v4 2.3GHz server at HCC, University of Nebraska-Lincoln. The CDP took almost 24 hours to identify

the CNSs among 17,996 orthologous syntenic gene sets of sorghum, setraia and rice. However, DiCE takes only 4.5 hours and 5 hours when it is tested with $p = 0$ and $p = 15$ respectively. Thus, it shows almost 5 times improvement of computational speed over CDP. The pairwise BLAST based comparisons in CDP are the most time consuming operations that makes it less efficient than DiCE.

3.5 Conclusion and Future Works

Comparative genomics approaches have been useful for studying the gene-regulatory elements of closely related species. The fundamental principle of comparative genomics is that the function-less DNAs change more rapidly as compared to functional DNAs. Many conserved noncoding sequences (CNSs) in the non-functional DNAs have been found to be transcriptional regulatory elements. Because of availability of genomes and annotations of many closely related species, it is important to identify the conserved regions across the species to study the functional properties of CNSs and gene-regulation. The CNSs in plants are much smaller than the CNSs in animals. Therefore, an efficient computational method to identify the accurate CNSs is desirable.

Most of the existing alignment based methods are neither computationally efficient nor sensitive to smaller CNSs found in plant species. In this paper, we present an algorithm that is based on generalized suffix tree and maximum weighted path in a DAG. Our algorithm, called DiCE, is not based on multiple pairwise alignments. The DiCE identifies the CNSs from the maximal repeat fragments i.e. MEMs found using a generalized suffix tree of sequences. It discovers the exact matches CNSs as well as the CNSs with a given mismatch rate. We tested our algorithm to identify the CNSs on set of 17,996 genes of six grass species. The experimental results show

that DiCE is 5 times faster than CDP, the best existing CNS discovery pipeline and it identifies almost 500 more CNSs. The accuracy and sensitivity of the results are evaluated using a permutation test. The results are further validated by comparing with previously known DHS sites of rice seed and callus that are related to CNSs.

In future, we plan to study scalability of the algorithms to a large number i.e. the order of thousands of plant species.

Chapter 4

Identifying conserved non-coding elements using min-wise hashing

Publication:

- Sairam Behera, Jitender S. Deogun, Etsuko N. Moriyama, 2020, “MinCNE: Identifying Conserved Non-Coding Elements using. Min-Wise Hashing”, *In: Advances in Computer Vision and Computational-Biology*. Arabnia HR, Deligiannidis L, Shouno H, Tinetti FG, Tran Q, eds. Springer. (In Press)

4.1 Introduction

Non-coding regions such as introns and intergenic regions of a genome are usually more divergent and exhibit higher molecular evolutionary rates compared to exon regions. Conserved non-coding elements (CNEs) are the genomic regions that show unusually extreme conservation. These elements are mostly clustered around the genes and play important roles in regulating the transcription process [113]. These elements (or regions) are also referred as conserved non-coding sequences (CNSs), ultraconserved elements (UCEs) or ultraconserved non-coding elements (UCNEs). The identification of CNEs in animal and plant genomes poses different challenges due to their sizes. CNEs in plants are shorter (15~50 bp) compared to animal CNEs

(≥ 100 bp) [130, 138]. The two major approaches that have been used to identify CNEs are alignment-based and alignment-free methods. The approaches can also be classified based on pairwise or multiple sequence comparison. Pairwise methods work on exactly two input sequences. Therefore, it requires multiple pairwise operations to process more than two sequences. The use of more than two sequences at once also poses challenges for scalability and computational efficiency compared to pairwise operations. Probabilistic data structures and approximate methods are often used to address scalability challenges.

The alignment-based approaches for CNE identification employ either pairwise or multiple sequence alignment methods. The most commonly used alignment tools are BLAST [4], QUOTA-ALIGN [131], LASTZ [57], BLASTZ [122], and MULITZ [20]. In some studies, CNEs are identified by manual or automated curation of BLASTN results [130, 138]. Others used global alignment with sliding window [10] or whole-genome alignment [59] to identify CNEs.

Among the first alignment-free tools that were developed for finding CNEs in plants were STAG-CNS [74] and DiCE [11]. STAG-CNS used suffix-tree based indexing and a directed acyclic graph to discover order-aware exact matched CNEs in various grass species, where the minimum length of CNEs can be as short as 8 bp. DiCE is the extension of the STAG-CNS approach, where the exact-matched CNEs are further processed in a brute-force manner allowing a given percentage of mismatches. CNEFinder [7] identifies CNEs longer than 200 bp in animal genomes. It finds the maximal exact matches (MEMs) between two given sequences using k -mer based methods and then extends the MEMs to produce the CNEs. CNEFinder employs a pairwise approach, whereas STAG-CNS and DiCE are designed to work with multiple sequences simultaneously. The approach used in DiCE for finding CNEs with mismatches is not computationally efficient due to its brute-force nature. This

motivated us to design an efficient algorithm for the CNE identification problem.

In this study, we propose an efficient alignment-free method, called MinCNE. MinCNE identifies the CNEs conserved among more than two sequences with user-defined constraints. Instead of finding exact-matched (identical) k -mers, our method clusters the similar k -mers with a given mismatch rate using min-wise hashing (min-hash) and locality-sensitive hashing (LSH). These two hashing approaches are highly efficient for clustering the elements using the Jaccard similarity measure. It ensures that the CNEs with the user-defined similarity are grouped together. MinCNE can identify CNEs as short as 100 bp. With its fast and efficient resource usage as well as the user-customizable similarity threshold, MinCNE is expected to contribute to discovery of more CNEs from a wide range of organisms.

4.2 Materials and Methods

Given a set of sequences, $S = \{s_1, s_2, \dots, s_n\}$, a minimum CNE length, k , and a similarity threshold, θ , MinCNE uses minhash and LSH strategies to identify all CNEs of length $\geq k$ present in all input sequences. The algorithm used in MinCNE is given in Algorithm 3 and the flowchart summarizing the MinCNE process is shown in Fig. 5.2. MinCNE is written in C++ and distributed under the GNU Public License (GPL). The source codes and relevant documents are freely available at <https://github.com/srbehera/MinCNE>.

4.2.1 Minhash signatures

The minhash is useful when the Jaccard similarity needs to be measured for large data sets. The Jaccard similarity index, which is also known as Intersection over Union, is used to represent the similarity between two sets. The similarity index between the

Algorithm 2: Identify CNEs using minhash and LSH

```

1 Set of sequences  $S = \{s_1 \cdots s_n\}$ ,  $k$ -mer size, number of hash functions  $N$ ,
   $q$ -gram size, band size  $b$ , similarity threshold  $\theta$ , hash functions  $H$  minHash,
  edlib, LSH List of CNEs with start and end positions Initialize cluster set
   $C \leftarrow \phi$  (empty)
2 Initialize list  $L \leftarrow \phi$  (empty)
  /* process all but the first sequence */
3 for each sequence  $s_i \in \{s_2, s_3, \dots, s_n\}$  in the  $S$  do
4   extract all  $k$ -mers and put in set  $K_i$ 
5   for each  $k$ -mer  $k_j \in \{k_1, k_2, \dots, k_{|s_i|-k+1}\}$  in  $K_i$  do
6     /* generate minhash signature of  $k$ -mer by using  $q$ -grams
7       and set of hash functions  $H$  */
8      $min\_sketch \leftarrow \text{minhash}(k_j, q, H)$  // set of  $N$  64-bit integers
9      $r \leftarrow \frac{N}{b}$ 
10     $B \leftarrow \text{LSH}(min\_sketch, b, r)$  // set of bucket ids
11    Assign sequence  $k_i$  in buckets whose ids are in  $B$ 
12  end
13 end
  /* process the first sequence */
14 extract all  $k$ -mers and put in set  $K_1$ 
15 for each  $k$ -mer  $k_j \in \{k_1, k_2, \dots, k_{|s_1|-k+1}\}$  in  $K_1$  do
16   create a cluster  $C$  and assign  $k_j$  to it
17    $min\_sketch \leftarrow \text{minhash}(k_j, q, H)$ 
18    $r \leftarrow \frac{N}{b}$ 
19    $B \leftarrow \text{LSH}(min\_sketch, b, r)$  // set of bucket ids
20   for each bucket_id  $b_k$  in  $\{b_1, b_2, \dots, b_r\}$  in  $B$  do
21      $B \leftarrow$  Bucket with id  $b_k$ 
22     if  $B$  has  $k$ -mers from  $n - 1$  sequences then
23       for each  $k$ -mer  $k_u$  in  $B$  do
24          $per\_id \leftarrow \text{edlib}(k_j, k_u)$ 
25         if  $per\_id \geq \theta$  then
26           Put  $k_u$  into  $C$ 
27         end
28       end
29     end
30   end
31   Process the cluster  $C$  to keep the one  $k$ -mer from each sequence with
32   highest  $per\_id$  score with  $k_j$ 
33   if  $C$  has  $n$  elements then
34     Add  $C$  to  $C$ 
35   end
36   Clusters that contain consecutive  $k$ -mers are merged and put into CNE
37   list  $L$ 
38 end
39 return  $L$ 

```

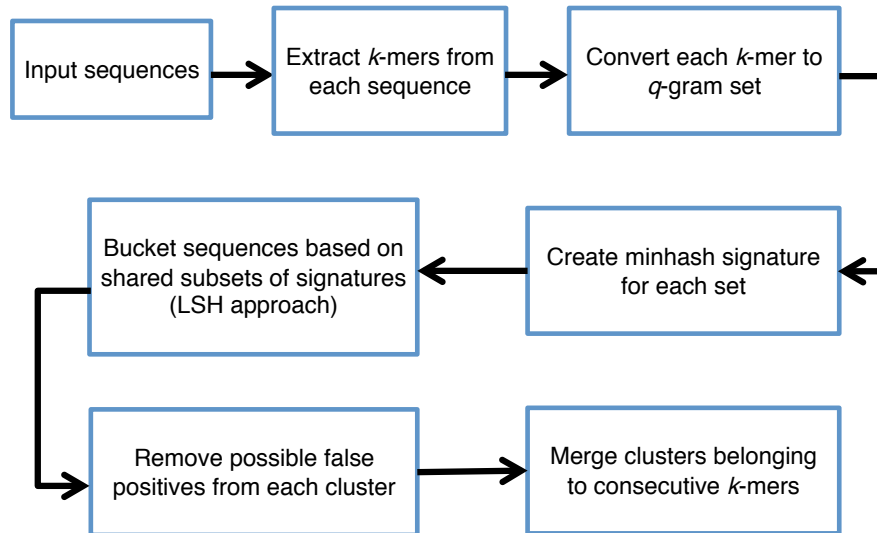


Figure 4.1: Flowchart of MinCNE: k -mers extracted from each sequence are converted first to q -gram sets and next to minhash signatures. LSH creates the initial cluster. k -mers in each cluster are compared to remove potential false positives. Clusters are merged to generate the final set of CNEs.

two sets X and Y is given as:

$$J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} \quad (4.1)$$

The earliest work of estimating the Jaccard similarity between sets of any sizes using minhash is found in [22]. A set of hash functions are used to convert each of the two sets into a minhash signature as follows. Each independent hash function generates a hash value for each element of the set. The minimum value among all hash values generated by the same hash function across all elements of the set is collected as an element of the minhash signature. With N independent hash functions, the minhash signature is a set of N elements corresponding to these minimum values. Therefore, the size of a minhash signature depends on the number of the independent hash functions used and independent of the size of the original set.

Let h_{min} be a minhash function and the collection of minimum hash values of the sets X and Y be $h_{min}(X)$ and $h_{min}(Y)$, respectively. It is shown that the probability of the two minimum hash value sets being equal is the Jaccard similarity of the sets X and Y [22]:

$$P(h_{min}(X) = h_{min}(Y)) = J(X, Y) \quad (4.2)$$

Given the minhash signatures of the two sets, both with the size N , let z be the number of minhash values that are shared, i.e. $|h_{min}(X) \cap h_{min}(Y)|$. Then an unbiased estimate of the Jaccard similarity is obtained by dividing z by N [157].

For MinCNE, the input sequences are pre-processed by enumerating all k -mers from each sequences. Each k -mer is further tokenized by extracting all possible q -grams (q -mers, $q \ll k$). A hash function converts each token into a 64-bit integer, and the minimum among them is selected. This process is repeated several times with different hash functions. With N different hash functions, a 64-bit integer vector of size N is generated for each k -mer. This vector is the minhash signature for the k -mer. This process is equivalent to selecting N random q -grams from a k -mer. It is expected that if two k -mers are similar, they share many q -grams. The Jaccard similarity between two k -mers, i.e. the proportion of shared q -grams between them, can be approximated by comparing the signatures as discussed above. However, the pairwise comparison of every possible k -mers is still computationally expensive. Therefore, the LSH algorithm is used to cluster the k -mers with similarities.

4.2.2 LSH-based clustering

LSH indexing was first developed for a general approximate nearest-neighbor search problem in high-dimensional spaces [65]. A family of hash functions are chosen in such a way that the collision probabilities of those hash functions are always high for

Table 4.1: Generation of minhash signatures for two k -mer sequences S_1 and S_2 .

5-grams	S_1					5-grams	S_2				
	H_1	H_2	H_3	H_4	H_5		H_1	H_2	H_3	H_4	H_5
caagt	11	67	9	89	56	cagtc	18	12	59	97	29
aagtc	98	53	16	9	67	agtct	88	32	99	7	23
agtct	88	32	99	7	23	gtcta	2	78	52	92	50
gtcta	2	78	52	92	50	tctag	10	7	88	70	39
tctag	10	7	88	70	39	ctagt	13	14	96	89	5
ctagt	13	14	96	89	5	tagta	58	61	28	1	15
tagta	58	61	28	1	15	agtag	76	58	43	11	52
agtag	76	58	43	11	52	gtaga	92	62	14	3	6
gtaga	78	42	59	82	31	tagat	19	39	23	88	97
tagac	66	71	45	92	4	agatg	86	10	77	31	3
agacg	32	38	93	72	21	gatga	44	96	29	9	47
gacga	69	51	94	6	7	atgac	29	52	75	95	53
acgac	73	71	99	88	14	tgact	20	23	9	82	88
cgact	92	75	8	62	22	gactt	59	40	86	18	28

H_1, \dots, H_5 are hash functions and hash values shown in red are the minimum values of each column.

similar inputs and low for dissimilar inputs. A formal definition of LSH functions is found in [65]. The minhash function h_{min} belongs to the family of LSH functions for the Jaccard distance, as the probability of collision is equal to the Jaccard similarity.

A minhash LSH index is built as follows. Once the minhash signatures are generated from all input data (e.g., sequences each represented by a set of q -grams), all signatures are divided into b bands of a fixed size r . If the minhash signature size is N , then $N = b * r$. We define the hash function H , which generates a bucket signature B_i for the i^{th} band by taking minhash signature values from positions $i * 1$ to $i * r$ as input:

$$B_i = H(h_{min,i*1}, h_{min,i*2}, \dots, h_{min,i*r}) \quad (4.3)$$

The bucket signature B_i maps a band in a signature to a bucket so that minhash signatures with the same bucket signature on the band i are mapped to the same

bucket. The minhash signatures of the two sets compared are mapped to buckets using the same set of hash functions. The two sets are considered to be the candidates of a similar pair if the signatures map to at least one same bucket. The time complexity of searching the candidate pairs using the LSH algorithm depends on the number of minhash functions (the minhash signature size) and is sub-linear with respect to the total number of sets in the search space. Let j be the Jaccard similarity between the sets X and Y , i.e. $j = J(X, Y)$. The probability that X and Y are the candidate pair is calculated as:

$$P(j|b, r) = 1 - (1 - j^r)^b \quad (4.4)$$

While the sets that meet a given Jaccard similarity threshold should have a high probability of becoming a candidate pair, those that do not meet the threshold should have low probabilities of becoming candidate pairs. The parameters such as the number of the bands b and band size r need to be adjusted to achieve these requirements.

In MinCNE, the LSH algorithm is used for clustering of k -mers as follows. The minhash signature of each k -mer is broken into a series of bands. A hash function is generated for each band and this becomes the bucket id. The index of the k -mer is put into this bucket. If the signatures of two k -mers share a band, then their indices (start positions in the sequences) will be found in the same bucket. The chances of finding the two similar k -mers in the same bucket increases with the increase in the number of bands.

How a pair of k -mers are compared using the minhash signatures and LSH-based clustering is shown in the following example. Consider the following two k -mers where $k = 18$:

S_1 : caagtctagtagacgact

S_2 : cagtctagtagatgactt

Each k -mer is first converted into a q -gram set that contains every possible q -grams of the k -mer. Setting both q and the minhash signature size N to 5, five hash functions (H_1, \dots, H_5) are used to generate five hash values as illustrated in Table 4.1. The minhash signature of each k -mer is the vector containing the minimum value from each hash function (shown in red in Table 4.1). Thus the minhash signatures of the above two k -mers will be given as:

$$Sig(S_1) = \langle 2, 7, 8, 1, 4 \rangle$$

$$Sig(S_2) = \langle 2, 7, 9, 1, 3 \rangle$$

With the number of bands $b=5$ and band size $r=1$, we have the following clusters:

$$C_1: \langle 1, 2 \rangle, \quad C_2: \langle 1, 2 \rangle, \quad C_3: \langle 2 \rangle, \quad C_4: \langle 1 \rangle, \quad C_7: \langle 1, 2 \rangle, \quad C_8: \langle 1 \rangle, \quad C_9: \langle 1 \rangle$$

4.2.3 CNE identification

Once the clusters are identified, the next task is to identify the CNEs. A CNE is required to be present in all given sequences. Therefore, we first discard the clusters that do not contain k -mers from all sequences. In the above example, clusters C_3 , C_4 , C_8 , and C_9 will be discarded. The k -mers clustered by LSH are likely to have higher Jaccard similarity scores than those that are not clustered. To ensure the similarity between every pair is greater than or equal to the given threshold (θ), sequences of each k -mer pairs in each cluster is compared. Edit distances are calculated using edlib, a lightweight and fast C++ library [159]. The clusters containing consecutive

k -mers are merged and extended until the similarity score drops below the threshold. For example, if we have the following five clusters containing the start positions of 200-mers in the original sequence:

$C_i: <10456, 39898, 78907 >$

$C_j: <10457, 39899, 78908 >$

$C_k: <10458, 39900, 78909 >$

$C_l: <10459, 39901, 78910 >$

$C_m: <10460, 39902, 78911 >$

these clusters can be merged to generate a CNE of length 205. The start and end-positions of three CNEs identified are as follows:

$<10456-10660, 39898-40102, 78907-79111 >$

In this study, the size of the minhash signature (N) was set to 50 generated by 50 minhash functions. Other settings used include: $q = 13$, $b = 25$, $r = 2$, and $k = 200$. The threshold for pairwise sequence similarity (θ) was set to 95%. These parameters were found to be optimal for the test sequences used in this study and no false positives were produced. Depending on the target CNEs, k -mer size can be set as short as 100.

4.2.4 Benchmark dataset

UCNEbase [37] is a publicly available database that contains the information about UCNEs of 18 vertebrate genomes. There are currently ~ 4300 CNEs present in the database. Almost half of them are from intergenic regions and others are from either

intron or untranslated exon regions. The non-coding regions of the human genome that exhibits more than 95% identity with chicken sequences are considered to be UCNEs. The minimum length of UCNE is 200 bp. To the best of our knowledge, this is the most recently updated database for CNEs of the human genome. We therefore chose this database to be the current benchmark. It should be noted that no independent verification has been performed for any CNEs under the definition of UCNEbase. We selected human intergenic UCNEs found in the following five gene regions: ZEB2, TSHZ3, EBF3, BCL11A, and ZFH4. From 1 Mbp regions both upstream and downstream of the coding regions of the five genes, UCNEbase recognized 271 UCNEs in total. The genomic sequences from five vertebrate species included: human (hg19), mouse (mm10), opossum (monDom5), chicken (galGal3), and zebra finch (taeGut1).

4.2.5 Performance evaluation

All experiments were run on the CentOS Linux server with Intel(R) Xeon(R) CPU E5-2630 v4 at 2.20GHz. All programs were run on a single core.

An identified CNE is considered to be a true positive if sequences identified from all species used have more than 95% sequence identity with the benchmark CNE sequences. The CNE-finding performance was examined using following metrics:

- *TP* (true positives): the number of identified CNEs that are present in UCNEbase
- *FP* (false positives): the number of identified CNEs that are not found in UCNEbase
- *FN* (false negatives): the number of CNEs that are present in UCNEbase, but are not identified by the tool

- Precision or positive predictive value: $\frac{TP}{(TP+FP)}$
- Recall or true positive rate: $\frac{TP}{(TP+FN)}$

Note that we did not include negative data; hence no true negative was counted. CNEs were identified from the 1 Mbp upstream and downstream of each gene region. Some of these test regions overlapped with exon regions of neighboring genes. Since our benchmark dataset was derived from UCNEbase, which does not recognize any conserved sequences from exon regions, any CNE candidates identified in these regions were excluded from the analyses.

We compared the performance of MinCNE with CNEFinder. CNEFinder works only on a pair of sequences for CNE identification. Therefore, direct performance comparisons were performed using only human and chicken sequences. The time and space efficiency of CNEFinder for multiple sequence comparisons was estimated based on the number of operations needed to be executed.

4.3 Results and Discussion

4.3.1 CNE identification performance

We first compared the CNE-finding performance between MinCNE and CNEFinder. Because CNEFinder can be used only for pairwise comparisons, we limited the comparison for human and chicken sequences.

As shown in Table 4.2, both MinCNE and CNEFinder were able to identify most of the benchmark CNEs. Out of 271 CNEs, MinCNE and CNEFinder missed 7 and 9 CNEs in total, respectively. MinCNE was able to find all 44 CNEs for EBF3 and 80 out of 81 CNEs for TSHZ3. CNEFinder identified all CNEs for TSHZ3, but missed only one for EBF3. For ZFH4, both MinCNE and CNEFinder failed to identify

Table 4.2: Comparison of MinCNE and CNEFinder using human and chicken dataset

Gene	UCNEbase	TP		FN		Recall		N/A*	
		MinCNE	CNEFinder	MinCNE	CNEFinder	MinCNE	CNEFinder	MinCNE	CNEFinder
ZEB2	63	61	61	2	2	0.97	0.97	25	21
TSHZ3	81	80	81	1	0	0.99	1	16	16
EBF3	44	44	43	0	1	1	0.98	25	24
BCL11A	32	31	29	1	3	0.97	0.91	15	15
ZFHX4	57	54	54	3	3	0.95	0.95	20	18

*All these sequences were found in the exon regions of other genes and not counted for the performance analysis.

the same set of three out of 57 CNEs. The recall values were $\geq 95\%$ and $\geq 91\%$ for MinCNE and CNEFinder, respectively. About a half of the FNs were the same CNEs missed by both MinCNE and CNEFinder (4 of 7 and 4 of 9, respectively). As noted above, the 1 Mbp test regions included some exon sequences of other genes. Both MinCNE and CNEFinder found CNE candidates in these regions (shown as N/A in Table 4.2), with many of them from the same regions. Neither of the tools produced FPs from any gene regions. Thus, the precision values were one for all tests.

Unlike CNEFinder, MinCNE can identify CNEs among multiple sequences at once. To demonstrate this capability, we performed the CNE identification using MinCNE with the sequences from three, four, and five species. The performance was exactly the same as shown in Table 2 (the same numbers of TPs and FNs were identified). This was expected because the two species compared in Table 4.2 are human and chicken, which are the most divergent pair of species among those compared (human, mouse, opossum, zebra finch, and chicken). Therefore, even when more species were included, since they were more closely related to either human or chicken, it did not increase the number of CNEs identified. It should also be noted that MinCNE identified all corresponding CNEs from additional species without exception. This demonstrates that MinCNE can efficiently and accurately identify CNEs from multiple genomes.

4.3.2 Time and space usage

The time and space usage of MinCNE was examined using different numbers of sequences. For CNEFinder, usages for more than two sequences were estimated based on the number of operations required in either serial or parallel execution. For example, if there are 10 datasets used in an experiment, there will be 45 pairwise operations. For serial executions, the estimated time for CNEfinder will be 45 times of a single execution. If all the executions are done in parallel, then the estimated time for the completion of all executions will be same as a single execution.

CNEFinder was faster than MinCNE when only two sequences were used (3 minutes by CNEFinder and ~ 9 minutes by MinCNE; Fig. 4.2a). The LSH-based clustering stage of MinCNE produces many clusters including many redundant ones. The processing of those clusters is the time consuming step of MinCNE. The minhash signature generation and initial clustering time increases sub-linearly with increase of the number of sequences. However, the time for identifying CNEs from the clusters decreases with increase in the number of sequences. This is because if a cluster does not contain a k -mer that is found in every sequence, it is eliminated. Therefore, the time usage with MinCNE did not increase with the number of sequences. In contrast, CNEFinder will require to be run ten times longer to compare five sequences. If these ten operations are executed serially, as shown in Fig. 4.2a, CNEFinder will take almost 30 minutes. CNEFinder will also require additional time for the post-processing of results from all pairwise runs.

The memory consumption of MinCNE was comparable to CNEFinder. Both tools used approximately 3 gb of RAM. As shown in Fig. 4.2b, the space usage increased only gradually when more sequences were used with MinCNE. Similar to computational time analysis, CNEFinder will need to be run ten times either serially

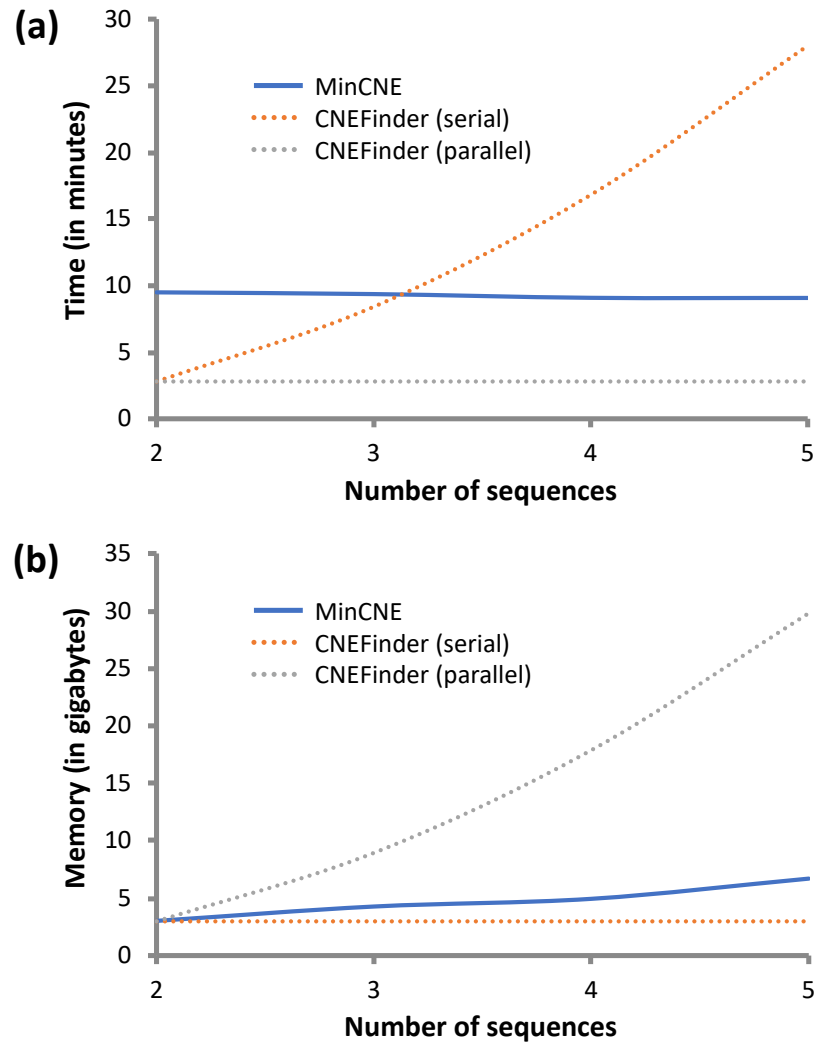


Figure 4.2: Time (a) and space (b) usage of MinCNE and CNEFinder: For CNEFinder, only the time and memory amount used for the two-sequence comparison was based on the actual observation. Other data were estimates.

or parallelly for five sequences. For parallel executions, the space usage for CNEFinder will increase by ten times. Additional space is also needed for CNEFinder for post-processing of pairwise results.

Although the current implementation of MinCNE does not support multi-threading, this will be added in the future version of MinCNE. With multi-threading, the advantage of using time and space efficient MinCNE is expected to be even more significant.

4.4 Conclusion

Minhash has been used in various bioinformatics applications especially for analyzing large datasets. We applied this technique in MinCNE, a new computationally efficient CNE finder. MinCNE does not require whole genome alignment nor multiple pairwise alignments for generating indices for the given sequences. Unlike other CNE-finding tools, MinCNE can work on more than two sequences at once. Our previous tool STAG-CNS found only exact matched CNEs [74]. This requirement was relaxed in DiCE [11]. However, DiCE was not computationally efficient especially with multiple long sequences. With MinCNE, we addressed these challenges. MinCNE is also flexible and the sequence identity threshold can be customized. Although CNEFinder uses the k -mer based technique and computationally efficient, it works only on two sequences at once. It requires multiple pairwise operations if multiple sequences need to be analyzed. Currently available CNE databases such as Ancora [41], CEGA [39], cneViewer [111], CONDOR [148], UCBase [85], UCNEbase [37], and VISTA [142], are mostly static and not updated regularly. MinCNE, with its computational efficiency, high sensitivity, as well as the flexibility, will be useful for studies in large-scale comparative genomics. The approximation techniques used by minhash and LSH can be further improved to reduce both space and time efficiency.

Chapter 5

Isoform clustering using minhash and locality-sensitive hashing

Publication:

- Sairam Behera, Jitender S. Deogun, and Etsuko N. Moriyama. 2020. MinIsoClust: Isoform clustering using minhash and locality sensitive hashing. *In Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (BCB '20)*. Association for Computing Machinery, New York, NY, USA, Article 64, 1–7.

DOI:<https://doi.org/10.1145/3388440.3412424>

5.1 Introduction

Using next-generation sequencing technologies, it is now easy to perform RNA sequencing (RNA-seq) for gene expression analysis and transcriptom assembly. However, especially in eukaryotes, a large proportion of genes are transcribed into multiple forms of transcripts (isoforms) through alternative splicing events. As shown in Fig 5.1, isoforms, although derived from a same gene, can code different protein sequences and hence can function differently. Differential expression of isoforms among tissues or developmental stages also contributes to increased complexity in eukaryotic tran-

scriptomes and proteomes. It has been reported that 95% of multi-exon genes in human undergo alternative splicing patterns[107]. Furthermore, more than 50% of disease-causing mutations in the human genome is estimated to affect splicing [86]. Therefore, accurate identification and quantification of isoforms is important not only for understanding of the mechanisms of biological complexity, but also for biomedical application.

There are two main strategies in computational transcriptome assembly: genome-guided and *de novo* methods [143]. In genome-guided methods such as Cufflinks [136] and StringTie [112, 71], short RNA-seq reads are mapped against a reference genome and splice graphs can be built. Most of the *de novo* transcriptome assemblers construct *de Bruijn* graphs using k -mers (substrings of length k in a DNA sequence) during the assembly process. For both strategies, existence of isoforms affects the performance of transcriptome assembly, often generating fragmented contigs. With the arrival of third-generation sequencing techniques (e.g., PacBio), long-read sequencing can be used to obtain the full-length isoform sequences. While this approach can potentially eliminate the need of isoform assembly, in contrast to the short-read sequencing performed by the Illumina platform, long-read sequencing is known to be highly error prone. Using any of these strategies, as shown in Fig 5.1, assembled contigs derived from the same gene are expected to both share highly similar or identical sequence regions (shared exons) have other regions that are unique to each isoform. To identify the sets of potential isoforms, these partially highly similar sequences need to be clustered, and such methods need to be scalable to deal with a large number of contigs generated from complex transcriptomes.

The classical clustering techniques are useful when the overall identities between two sequences are used. However, clustering of the isoforms that are generated with various splicing patterns poses a major challenge to existing clustering techniques be-

cause sequences include both highly similar shared exons regions as well as dissimilar unique exons regions.

CD-HIT is the most often used sequence clustering tools , which was originally developed for protein sequence clustering and now is applicable for nucleotide sequences [80]. The greedy clustering approach with short-word filtering used with CD-HIT makes it computationally highly efficient. However, it is known to generate false negatives when the contigs are clustered at the isoform level [35]. MMseqs2/Linclud is another clustering method that is expected to run faster than CD-HIT [129]. The algorithm of MMseqs2/Linclud is based on shared k -mers between the sequences. isONclud is one of the recent clustering tool that is designed to cluster PacBio and nanopore data efficiently. The clustering method in isONclud uses shared k -mer and minimizer scheme [119]. Although MMseqs2/Linclud is not designed to work for isoforms, we included this tool as the results in [129] show that it is faster than CD-HIT. isONclud is also included as it works for transcriptome data.

To address the challenges related to scalability and accuracy of isoform identification, in this study, we developed a novel approach to cluster the transcript sequences potentially derived from isoforms, MinIsoClust. MinIsoClust makes use of minwise-hashing (minhash) technique to generate signatures for input sequences and locality-sensitive hashing (LSH) approach for initial clustering eliminating the requirement of pairwise comparisons of all input sequences. We further use efficient edit-distance computation tool and bloom-filter based approximation to find the containment of a sequence in another sequence. To test this new method, we generated four simulated datasets. Isoform clustering performance of MinIsoClust was compared against CD-HIT, isONclud, and MMseqs2/Linclud. MinIsoClust demonstrated more accurate isoform clustering for most of the datasets and maintained very high computational efficiency.

Algorithm 3: MinIsoClust isoform clustering

```

1 Set of sequences  $S = \{s_1 \cdots s_n\}$ , number of hash functions  $N$ ,  $q$ -gram size,
  band size  $b$ , similarity threshold  $\theta$ , hash functions  $H$ , shared length  $l_s$ 
  minHash, edlib, LSH Clusters each containing potential isoforms Initialize
  cluster set  $K \leftarrow \phi$  (empty)
2 Initialize list  $L \leftarrow \phi$  (empty)
3  $l_m \leftarrow$  length of shortest sequence
  /* process all sequences */
4 for each sequence  $s_i \in \{s_2, s_3, \dots, s_n\}$  in the  $S$  do
5   Initialize minhash signature set  $MH_i \leftarrow \phi$  (empty)
6    $l_i \leftarrow$  length of sequence  $s_i$ 
7    $t \leftarrow 2 \times \frac{l_i}{l_m}$ 
8   Generate  $t$  random positions between 0 and  $(l_i - l_m - 1)$ 
9   Extract subsequences of length  $l_m$  from these positions and put it in set  $F_i$ 
10  for each subsequence  $f_j \in \{f_1, f_2, \dots, f_t\}$  in  $F_i$  do
11    /* generate minhash signature of  $f_j$  by using  $q$ -grams and
12     set of hash functions  $H$  */
13     $min\_sketch \leftarrow$  minhash( $f_j, q, H$ ) // set of  $N$  64-bit integers
14    Add  $min\_sketch$  to  $MH_i$ 
15     $r \leftarrow \frac{N}{b}$ 
16     $buk \leftarrow$  LSH( $min\_sketch, b, r$ ) // set of bucket ids
17    Assign sequence  $s_i$  in buckets whose ids are in  $buk$ 
18  end
19 end
20 Set of isoform clusters  $I \leftarrow \phi$  (empty)
  /* Second pass of sequences */
21 For each sequence  $s_i$ , assign a flag  $g(s_i)$  and set it to zero
22 for each sequence  $s_i \in \{s_2, s_3, \dots, s_n\}$  in the  $S$  do
23   cluster  $K \leftarrow s_i$  /* Not yet added to any cluster */
24   if  $g(s_i) == 0$  then
25      $B \leftarrow$  sequences from its corresponding buckets
26     for each sequence  $s_u$  in  $B$  with  $g(s_u) == 0$  do
27       Add  $s_u$  to cluster  $C$  if it satisfies for any  $z \in C$ 
28       1. edlib( $s_u, z$ )  $\geq \theta$ 
29       2.  $s_u$  is contained in any  $z$ 
30       3.  $s_u$  shared at least  $l_s$  long sequence with any  $z$ 
31        $g(s_u) \leftarrow 1$ 
32     end
33   end
34   Add  $C$  to  $I$ 
35 end
36 return  $I$ 

```

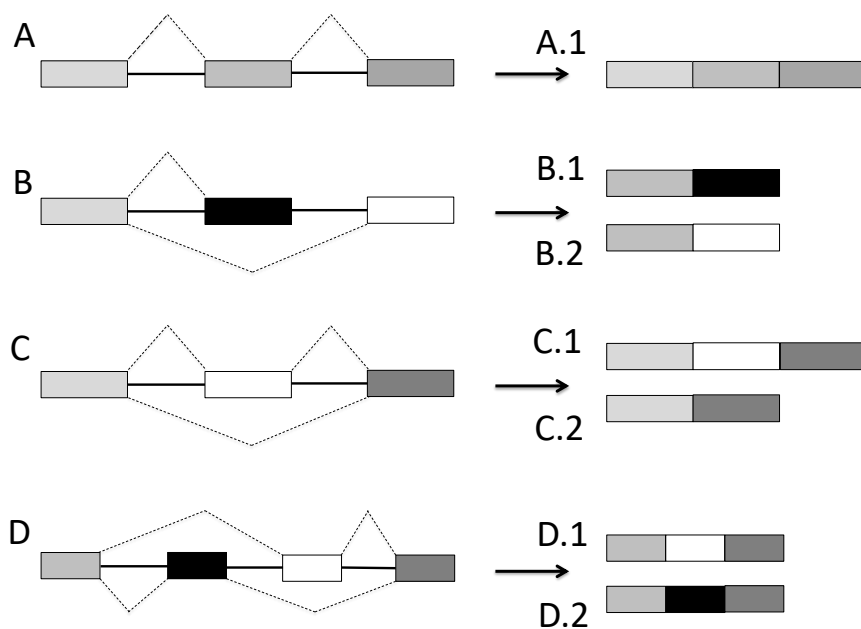


Figure 5.1: Various alternative-splicing events: Exons are represented by boxes, and introns connecting exons are represented by the solid lines. Dotted lines show the splicing events. Transcripts produced by various splicing events are shown on the right side. A-D illustrate the gene structures. A.1, B.1, ..., D.2 illustrate the transcripts produced after splicing events. While gene A produces only one type of the transcript (A.1), genes B, C, and D undergo alternative splicing events and produce more than one forms of transcripts (isoforms) as illustrated in B.1, B.2, ..., D.2.

5.2 Materials and Methods

5.2.1 Sequence comparison using minhash signatures

The Jaccard similarity is popularly used to assess the similarity between any two sets and defined as the ratio of the size of the intersection to the size of the union. For genomic sequences similarity, the Jaccard similarity score can be computed by converting each sequence into a set of smaller words, i.e. q -grams and then computing the intersection over union ratio for a pair of sequences. This requires extraction of all possible q -grams from all sequences. For genomic-level and high-throughput datasets, this is not an efficient approach. The minhash approach can be used to

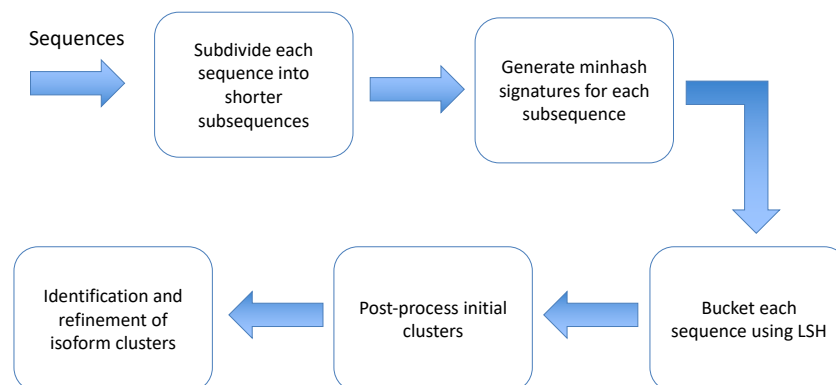


Figure 5.2: Flowchat of MinIsoClust: The sequences are first divided into subsequences. Each sequence is represented by a set of minhash signatures corresponding to its subsequences. LSH is used for bucketing each sequence based on the signatures. Post-processing of the initial clusters generated by LSH remove redundant clusters. Final refinement and identification of isoform clusters are done by pairwise processing of sequences and bloom filter.

approximate the Jaccard similarity score by converting the sequences into fixed-size integer sets chosen from hash values of randomly picked q -grams and then computing the intersection-over-union ratio for these sets instead [22].

However, while the minhash approach works well when the input sequences are of similar lengths, if the lengths of two sequences differ significantly and one sequence is either contained in another or if only segments of a longer sequence is similar to a shorter sequence, the minhash does not work well [31]. This is because many of the randomly picked q -grams from the longer sequence could come from the segments that are shared with the shorter sequence. Thus, even if a shorter sequence is completely contained in a longer sequence, the minhash-approximated similarity index becomes low. As shown in Fig 5.1, some isoforms share only a limited region, and all such potential isoform sequences need to be clustered. Therefore, we modified the minhash approach to ensure that it works well for sequences with varying lengths. This is called containment minhash approach and it has been studied for various applications

[124, 70]. Given two sequences X and Y where X is much longer than Y , we used the containment minhash approach as follows. Let L_X and L_Y be the lengths of X and Y , respectively, and $L_Y < L_X$. We divide the sequence X into several several subsequences of length L_Y . If the Jaccard similarity score between Y and one of the subsequences of X is higher than given similarity threshold, sequence Y is contained in X . We used this containment-sensitive minhash approach to identify isoforms.

5.2.2 MinIsoClust isoform-clustering strategy

The overall process of MinIsoClust is shown in Fig 2. MinIsoClust clustering is composed of three stages: 1) generation of minhash signatures of input sequences using minwise hashing (minhash) 2) the bucketing of potential isoform sequences using LSH, and 3) isoform clustering using bloom filter and pairwise sequence comparisons of sequences within the buckets. The algorithm is given in Algorithm 3.

Let $S = \{s_1, s_2, \dots, s_n\}$ be the set of sequences of varying lengths, and l_i and l_m be the lengths of sequence s_i and the shortest sequence of the set respectively. A q -gram is a subsequence of q consecutive characters with $q \leq 10$. Let r be the number of random positions, and H be the set of N hash functions $\{H_1, \dots, H_N\}$.

A minhash signature of each sequence is generated as follows. For each sequence s_i , $2 \times \frac{l_i}{l_m}$ subsequences, each with length l_m , are extracted from random positions. This ensures that there are at least two overlapping subsequences for each position. Each subsequence is then converted into a q -grams set. A set of N hash functions (H) is used to generate hash values for each q -gram in the set. Let v_i be the set of the hash values generated for all q -grams in the subsequence by a hash function H_i and $v_{i,min}$ be the minimum value in v_i . The minhash signature of the subsequence is the set of N minimum hash values, i.e. $\{v_{1,min}, \dots, v_{N,min}\}$. The collection of minhash signatures obtained from all subsequences becomes the minhash signature set of a

sequence. Thus, each sequence whose length is greater than l_m contains more than two minhash signatures.

Jaccard similarity scores between sequences and subsequences can be now calculated using the minhash signature sets. However, performing all pairwise comparisons using minhash signatures is still computationally expensive for a large number of sequences. To address this challenge, LSH-based bucketing method is used.

5.2.3 LSH-based bucketing

LSH is useful for hashing similar items into the same bucket with high probability [49]. For sequence similarity, the probability of two sequences are being similar increases with the increase in the number of shared elements in their minhash signatures. The LSH algorithm makes use of information about shared elements in minhash signatures to create buckets and put similar sequences into the same bucket. The minhash signatures are divided into b bands with each band containing $\frac{N}{b}$ elements. A hash value that is generated for each band becomes the signature of the bucket. If two signatures share the same band, then they should share the same bucket.

If the minhash signature of a shorter sequence Y and minhash signature of the subsequence of a longer sequence X share the same band, then both X and Y share the same bucket. Similarly, if the signatures of the subsequences of sequences Z and X share same band, then both X and Z share a bucket. As shown in Fig 5.1, these subsequence relationships represent shared exons between isoforms. The sequences that share similar subsequences are put into the same bucket. This effectively cluster potential isoforms. Note also that at this stage, each sequence can be put into several buckets as each subsequence can be put into at most b buckets.

Table 5.1: Distribution of numbers of isoforms in the four datasets

# isoforms per gene	Rice (16,894 ^a)	Soybean (17,226 ^a)	Arabidopsis (16,071 ^a)	Human (18,348 ^a)
1 ^b	16,613	15,782	9,109	8,481
2	267	1,228	1,915	2,393
3	14	171	514	795
4		31	168	288
5		10	41	77
6		2	17	30
7		2	3	17
8			2	6
9			0	2
≥10			1	9
% single gene	98.34%	91.62%	56.68%	46.22%
Total # transcripts	17,189	18,951	15,507	17,668
Expected isoform/gene	1.05	1.34	2.22	2.90

^a Total number of genes^b No alternative splicing

5.2.4 Identification of isoforms by clustering

The initial clusters generated in the bucketing stage contain redundant clusters and also some potential false positives. Therefore, we remove redundancy of the clusters and reduce potential false positives as follows. The buckets that share the same sequences are merged. Pairwise comparisons are done on the sequences in the merged bucket to check if they can form an isoform cluster. The `edlib` tool is used with the infix alignment mode to compute the edit distance between sequences [159]. The infix mode does not penalize the gaps at the start and end of the query, which is useful for finding if a sequence is completely contained in another. The threshold for the edit distance similarity is set to 95%.

To identify the isoforms such as those shown in Fig 5.1 B-D, bloom-filter based fast dictionary search is used. If the candidate isoforms I_1 and I_2 share all exons except some additional exons only in I_1 as shown in Fig 5.1C, all q -grams of I_2 is

present in I_1 . To quickly check the presence of all q -grams of I_2 in I_1 , a bloom filter is created for I_1 and filled with q -grams found in I_1 . All the q -grams of I_2 are queried against the bloom filter and if all of them pass the membership test, I_1 and I_2 are considered to be part of the same cluster. Similarly, if two candidate isoforms share some exons and each contain additional unique exons (as shown in Fig 5.1B and D), a bloom filter is also used to test for the membership of a subset of q -grams. The bloom filter [21] is a powerful probabilistic data structure that can be used for fast set-membership testing. It can perform pairwise processing very fast.

Table 5.2: Isoform clustering performance among the four methods

Dataset	MinIsoClust			isONclust			MMseqs2			CD-HIT		
	h	c	v	h	c	v	h	c	v	h	c	v
Rice	1	0.985	0.958	0.999	0.983	0.992	0.999	0.932	0.964	0.998	0.984	0.991
Soybean	1	0.900	0.947	0.683	0.968	0.801	0.663	0.871	0.753	0.687	0.999	0.815
Arabidopsis	0.958	1	0.979	0.990	0.996	0.993	0.975	0.956	0.965	0.958	0.995	0.979
Human	0.954	0.995	0.974	0.962	0.993	0.969	0.951	0.995	0.973	0.948	0.991	0.973

The scores in bold are the highest scores achieved by a tool for a particular dataset.

5.2.5 Benchmark datasets

We generated four simulated benchmark datasets based on four organismal model (Arabidopsis, Rice, Soybean, and Human). The data sources for the reference transcriptome of these four species are as follows: *Arabidopsis thaliana* Columbia (Col-0), human reference genome (HG38), *Oryza sativa* Japonica Nipponbare, *Glycine max* Williams 82. The simulated benchmark datasets were generated using Flux Simulator [152] with the four reference transcriptomes. The protocol to generate the benchmark datasets is detailed in [145]. These four datasets are selected based on the different levels of complexity in isoform distribution (shown in Table 5.1). Rice dataset is the simple dataset with no genes containing more than three isoforms. It also has the largest proportion of the genes with no isoforms (98.34%). Soybean dataset contains more than thousand genes with two isoforms (7.12%). Both Arabidopsis and

Human datasets have higher proportions of genes with multiple isoforms (43.32% and 53.78%, respectively). Human dataset has 9 genes with more than 10 isoforms. Thus we have a range of complexity in these datasets. These simulated datasets provided both ground-truth information as well as input data.

5.2.6 Performance evaluation

Clustering performance of MinIsoClust was compared against three other methods: CD-HIT, isONclust, and MMseqs. The following three metrics were used: homogeneity, completeness, and V -measure. Let $K = \{k_1, \dots, k_N\}$ be the set of clusters generated by any clustering algorithm, $G = \{g_1, \dots, g_M\}$ be the set of classes defined by the ground truth (benchmark data). n is the total number of input sequences, n_G is the number of sequences belonging to class G , n_K be the number of sequences belonging to cluster K , and $n_{G,K}$ be the number of elements of class G in cluster K . The homogeneity (h), completeness (c), and V -measures (v) are defined as follows [117]:

$$h = 1 - \frac{H(G|K)}{H(G)} \quad (5.1)$$

$$c = 1 - \frac{H(K|G)}{H(K)} \quad (5.2)$$

where $H(G|K)$ is the conditional entropy of the classes given the cluster assignments:

$$H(G|K) = - \sum_{i=1}^N \sum_{j=1}^M \frac{n_{g,k}}{n} \cdot \log \left(\frac{n_{g,k}}{n_k} \right) \quad (5.3)$$

and $H(G)$ is the entropy of the classes:

$$H(G) = - \sum_{i=1}^N \frac{n_g}{n} \cdot \log \left(\frac{n_g}{n} \right) \quad (5.4)$$

The V -measure is defined as the harmonic mean of homogeneity and completeness:

$$v = 2 \cdot \frac{h \cdot c}{h + c} \quad (5.5)$$

Homogeneity is a measure of the ratio of samples of a single class pertaining to a single cluster. The fewer the classes included in one cluster, the better. It ranges from 0.0 to 1.0. Completeness measures the ratio of the member of a given class that is assigned to the same cluster. The V -measure is calculated similar to the F -measure where precision and recall are combined. It is more comprehensive than using only either homogeneity or completeness.

One of the drawbacks of the above metrics is that those are not normalized with regards to random labeling. A complete random label is not always guaranteed to produce same values for homogeneity and completeness based on the numbers of the samples, clusters, and ground-truth classes. An adjusted index such as the Adjusted Rand Index (ARI) is used to address these issues. ARI measures the similarity of the two assignments, ignoring permutations and with chance normalization. Let a be the number of pairs of elements that are in the same set in G and in the same set in K , b be the number of pairs of elements that are in different sets in G and in different sets in K . The Rand index (RI) is given by:

$$RI = \frac{a + b}{G_2^{n_{samples}}} \quad (5.6)$$

where $G_2^{n_{samples}}$ is the total number of possible pairs in the dataset (without ordering).

The ARI is given by:

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]} \quad (5.7)$$

where $E[RI]$ and $\max(RI)$ are the expected and maximum RI . The formal

definitions and mathematical equations of above metrics can be found in [64].

5.2.7 Program execution

All experiemnts were run on the CentOS Linux server with Intel(Xeon CPU E5–2630 v4 at 2.20GHz using a single core. CD-HIT-EST, isONclust, and MMseqs2/Linclud were run with their default parameters. For MinIsoClust, the size of the minhash signature (N) was set to 200 and the number of bands (b) to be used in LSH bucketing was set to 40. The q -gram size was set to 5. The threshold for the pairwise sequence similarity (θ) was set to 95%. These parameters were found to be optimal for the test sequences used in this study. All evaluation metrics were computed using the `scikit-learn` library [109].

5.3 Results and Discussion

Table 5.3: Performance evaluation of isoform clustering using *ARI*

Dataset	MinIsoClust	isONclust	MMseqs2	CD-HIT
Rice	0.083	0.081	0.004	0.071
Soybean	0.331	0.003	0.008	0.01
Arabidopsis	0.954	0.826	0.224	0.003
Human	0.154	0.372	0.100	0.001

The scores in bold are the highest scores achieved by a tool for a particular dataset.

5.3.1 Isoform-clustering accuracy

The acuracy of all clustering tools were evaluated using the three metrics: homogeneity (h), completeness (c), and V -measure (v). The accuracy of the tools is directly proportional to the scores of these three metrics. Higher the h value, higher the chances that each cluster has member from a single class in the ground truth dataset.

Table 5.4: Number of singleton clusters generated by four methods

Dataset	Benchmark	MinIsoClust	isONclust	MMseqs2	CD-HIT
Rice	16,613	16,210	16,895	12,996	18,542
Soybean	15,782	14,876	11,997	12,585	17,734
Arabidopsis	9,109	9976	10,353	8682	13,724
Human	8,481	8643	8461	7311	9081

Similarly, a high c score indicates many of the members of a given class in the ground truth are grouped together in the same cluster. Table 5.2 shows that MinIsoClust outperformed all other tools in most of the cases. The Rice benchmark dataset is the least complex and has only 281 classes with more than one isoforms. All four tools had very high h score with this dataset. For the Rice and Soybean datasets, MinIsoClust had the perfect score (1) for the homogeneity indicating that for all identified isoform groups no other transcripts were incorrectly included. In contrast, especially for the Soybean dataset, other tools tended to cluster isoforms with incorrect transcripts. For the Arabidopsis dataset, one of the most complex datasets used in this study, MinIsoClust also showed the perfect score for the completeness indicating all isoforms were clustered together correctly..

In Table 5.3, the isoform clustering accuracy is compared using ARI . ARI ensures that there is no bias with respect to single accuracy measure. The scores of ARI ranges from -1 to 1 with 1 being the best. Note that while in Table 5.2, the differences in scores are not very large, in Table 5.3, the score differences are much more pronounced and MinIsoClust showed significantly better performance compared to others. To explore the performance difference in more detail, in Table 5.4, the numbers of singleton clusters predicted by each method is compared. It shows that while MMseqs2/Linclud tends to underestimate the number of singleton clusters, CD-HIT shows significant overestimation for all datasets. For all datasets, MinIsoClust shows the best estimates for singleton clusters.

5.3.2 Computational time and space usage

The computational time and space usage of MinIsoClust was examined and compared against the other three tools. As shown in Table 5.5, CD-HIT was the fastest among all methods. MinIsoClust was significantly faster than isONClust and MMseqs2/Linclud. Although for more complex Human dataset, the run-time was higher than for all other datasets with all tools, sometimes (e.g., for the Rice dataset), the run-time did not correlate with the complexity perceived simply based on the isoform numbers, indicating other factors affecting the process of isoform clustering.

Table 5.5: Run-time comparison among the four methods^a

Dataset	MinIsoClust	CD-HIT	MMseqs2	isONclust
Rice	17.22	2.46	323.99	85.97
Soybean	18.43	1.23	78.52	221.79
Arabidopsis	7.43	0.54	19.42	31.79
Human	19.57	3.24	569.78	319.03

^aThe run-time is shown in seconds.

For each dataset, the best performing method is shown in bold

The space usage of the four tools are given in Table 5.6. CD-HIT was the most space efficient among the four tools. isONclust consumed more memory than others for all the datasets. The memory consumption of MinIsoClust was comparable to CD-HIT. Similar to computational time efficiency, MinIsoClust was more space efficient than MMseqs2/Linclud and isONclust.

Table 5.6: Space usage comparison among the four methods^a

Dataset	MinIsoClust	CD-HIT	MMseqs2	isONclust
Rice	0.27	0.06	1.52	6.4
Soybean	0.1	0.05	0.32	3.30
Arabidopsis	0.13	0.58	1.47	3.59
Human	0.09	0.1	1.9	7.23

^aThe space usage is shown in GB.

For each dataset, the best performing method is shown in bold.

5.4 Conclusion

Clustering or cluster analysis is an important task for various bioinformatics applications. In this work, we proposed MinIsoClust, a minhash-based clustering tool for transcriptomes that contain isoforms. The minhash techniques have been proven to be efficient for estimating similarity in many applications that involve large datasets [76]. It has also been used for many bioinformatics applications especially for analyzing large-scale sequencing datasets. Our algorithm makes use of LSH-based bucketing from minhash signatures to cluster the isoform sequences. The conventional clustering tools such as CD-HIT is based purely on the sequence similarity and it was not expected to perform well for clustering transcripts that include alternative splicing events. Furthermore, for most of the clustering techniques, scalability is still a challenging issue. We addressed both of these challenges by integrating the containment component to the minhash approach and avoiding pairwise comparisons with the use of LSH bucketing. By using the four simulated benchmark datasets where the ground-truth isoform clustering is known, we could also conduct a fair clustering performance evaluation among the methods using various statistics.

Our results showed that MinIsoClust generated more accurate clusters than the other three tools. As expected, the computational time efficiency of MinIsoClust was significantly better than MMseqs2 and isONclust. While CD-HIT was more efficient than MinIsoClust, it was at the cost of generating false negatives i.e. missing isoforms in the clusters indicated by lower completeness scores. The space usage of MinIsoClust and CD-HIT was comparable and both performed better than isONclust and MMseqs2/Linclust. We plan to perform more analyses using larger datasets with more varied isoform distributions to investigate how these methods perform differently depending on the types of datasets.

One advantage of MinIsoClust is the flexible option to change the sequence similarity threshold (θ) for clustering shared sequences. By using lower threshold, MinIsoClust can be applied to both gene family clustering as well as error-prone third-generation sequence clustering. We plan to explore such applications of MinIsoClust in the future.

Chapter 6

New ensemble approach for improving transcriptome assembly

Publications:

- Sairam Behera, Adam Voshall, Jitender S. Deogun and Etsuko N. Moriyama, 2017, “Performance comparison and an ensemble approach of transcriptome assembly,” *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM '17)*, Kansas City, MO, pp. 2226-2228, doi: 10.1109/BIBM.2017.8218005.
- Adam Voshall, Sairam Behera¹, Xiangjun Li, Xiao-Hong Yu, Kushagra Kapil, Jitender S. Deogun, John Shanklin, Edgar B. Cahoon, Etsuko N. Moriyama, 2020, “A consensus-based ensemble approach to improve *de novo* transcriptome assembly”, *bioRxiv*; doi: <https://doi.org/10.1101/2020.06.08.139964>.
- Sairam Behera, Adam Voshall, Etsuko N. Moriyama. “Plant transcriptome assembly: review and benchmarking”, *Bioinformatics*, Brisbane: Exon publications; 2020. (In Press).

¹Helped in analysis for assembly using different identity thresholds

- Sairam Behera, Adam Voshall, Kushagra Kapil, Jitender S. Deogun, Etsuko N. Moriyama, 2020, “Minsemble: clustering based ensemble transcriptome assembly” (manuscript under preparation).

6.1 Introduction

A transcriptome is the entire set of transcripts in a cell. The content of a transcriptome varies between different types of cells (tissues) and between developmental stages. Understanding the content of transcriptomes and tracking their spatial and temporal differentiation is important when we study the mechanisms of, e.g., cellular differentiation, carcinogenesis, and gene regulation. RNA-sequencing (RNA-seq) is a transcriptome profiling technology that utilizes high-throughput next-generation sequencing. The majority of RNA-seq data are generated from the complementary DNAs (cDNAs) converted from messenger RNAs (mRNAs) by using the Illumina short-read sequencing platform [105, 93]. More recently, long-read and direct-RNA sequencing has also become available for RNA-seq using third-generation sequencing platforms [e.g., Pacific Biosciences (PacBio) and Oxford Nanopore (ONT)] [128].

RNA-seq provides a quantitative snapshot of a transcriptome of the cells at a given time point. RNA-seq data can be used to reconstruct transcriptomes and also to analyze differential gene expression and differential splicing of mRNAs. However, many challenges remain in assembling the transcripts correctly using the available assembly algorithms [141]. The sequencing errors and presence of repetitive sequences most often cause mis-assembly of transcripts. Shared exon regions and different expression levels among alternatively spliced transcripts (isoforms) make the identification and quantification of genes and isoforms challenging for transcriptome assembly and quantification tools [62]. For many plant species, polyploidy adds another level of

complexity for transcriptome assembly. The high sequence similarity among sub-genomes, among duplicate genes, as well as among isoforms all makes the *de novo* transcriptome assembly a significant challenge [55, 144].

In the following sections, we will first review three transcriptome assembly strategies: genome-guided, *de novo*, and ensemble. Next we will describe how the transcriptome assembly performance can be evaluated. We will discuss the advantage of using simulated benchmark data instead of actual data and outline how such simulated benchmark transcriptome datasets can be generated. Finally, we will demonstrate how transcriptome assemblies generated from different methods can be compared and how the transcriptome assembly quality can be evaluated using simulated plant transcriptomes with varied complexity.

6.2 Transcriptome Assembly Strategies

Transcriptome assembly is a process of reconstructing the complete set of full-length transcripts from RNA-seq data, which often include tens of millions of short-read sequences. Genome assembly methods cannot be used for transcriptome assembly due to drastically varied sequencing depth among transcripts (due to gene-expression variation), strand-specific experiments with RNA-seq, and existence of isoforms. For transcriptome assembly, genome-guided or reference-based assembly methods are preferred when a high-quality reference genome is available [93, 143]. *De novo* or reference-free transcriptome assembly methods do not require reference genomes. These methods are particularly useful for non-model organisms where often high-quality reference genomes are not available [51, 63, 89].

6.2.1 Genome-guided approach

The genome-guided approach of transcriptome assembly makes use of a genome sequence while reconstructing the transcripts [44]. These approaches first map the sequenced reads to the reference genome using a splice-aware aligner such as TopHat2 [67], HISAT2 [68] or STAR [38]. The mapping information is then used to construct a graph that represents the splice junction of the transcripts (splice graph). The final transcripts are extracted by traversing the graph. Bayesemblem [92], Cufflinks [135], StringTie [112], and Scallop [123] are some examples of genome-guided assembly tools that have been used extensively. To handle the presence of introns in the genome, the aligners take splice-junction sites into consideration and allow split-mapping where one part of a read is mapped to one exon and another part to another exon. One issue with using short reads is that they can be mapped to multiple locations in the genome due to the existence of repetitive sequences or highly similar duplicated genes. The read-mapping strategies used by different aligners handle such ambiguities differently [104]. The techniques used to construct the graph and the contig sequences from the mapping information are also different among the methods. Selection of aligners and assembly methods, therefore, has a significant impact on the assembly results. The availability of a high-quality reference genome is also necessary for accurate assembly. If the read sequences and the reference genome are not from the same strain of the same species, the resulting divergence in the read and reference sequences could also cause assembly mistakes.

Cufflinks is one of the most widely used genome-guided transcriptome assemblers [135]. It can be used not only to assemble transcripts but also to estimate their abundance and to test differential expression. Cufflinks constructs an overlap graph based on the alignments of the overlapping reads on the genome. Transcripts are

identified by traversing the minimal paths that cover all alignments in the graph (each path represents a different isoform). Since Cufflinks performs transcriptome assembly and expression-level estimation separately, it does not consider transcript abundance when finding the minimal set of transcripts. StringTie simultaneously assembles transcripts and estimates their expression levels [112]. From the clusters of reads mapped to the genome, it creates a splice graph for each cluster. It then traverses the splice graph to construct transcripts. For each transcript, it creates a flow network to estimate its expression level using an optimization technique known as the maximum flow algorithm. This information is iteratively used to update the splice graph. Scallop, a more recent genome-guided tool, also creates a splice graph from the clustered reads mapped on the genome [123]. It preserves phasing paths using the reads that span more than two exons. By iteratively decomposing each splice graph, it reduces false transcripts. By incorporating phasing information, Scallop achieves improved assembly of multi-exon transcripts and lowly expressed transcripts.

6.2.2 *De novo* approach

The *de novo* approach of transcriptome assembly reconstructs transcript sequences from short reads without using a reference genome. Most of the *de novo* transcriptome assembly techniques use the de Bruijn graph based on k -mers [32], which include Trinity [52, 58], IDBA-Tran [29], SOAPdenovo-Trans [149], and rnaSPAdes [24]. A k -mer of a sequence is a subsequence of length k , i.e., k consecutive nucleotides. During the assembly process, each sequence is decomposed into all possible fixed-size k -mers. The nodes or vertices of a de Bruijn graph are represented by the k -mers. An edge is created between two nodes if the corresponding k -mers have a suffix-prefix overlap of length $k-1$, i.e., the last $k-1$ nucleotides of one k -mer exactly match with first $k-1$ nucleotides of the other k -mer. Two consecutive k -mers of a sequence, therefore,

can be represented as two nodes with an edge between them. Thus, a de Bruijn graph represents a set of reads as each read induces a sequence of edges that joins a sequence of vertices, i.e., a path. If two read sequences share a subsequence, then a common path is induced in the graph. If two read sequences have a suffix-prefix overlap, then a single path is induced for both sequences. After a de Bruijn graph is constructed, different paths are traversed to generate the putative transcripts. Note that if the reads are derived from highly similar (but not identical) sequences, they create isolated nodes and loops, which affects the accuracy of the graph construction. Sequencing errors can also cause false k -mers (those containing erroneous nucleotides) to participate in the graph construction by creating false nodes. The false nodes either break the path or creates a false path if overlapped with another k -mer.

For de Bruijn graph-based assembly methods, the choice of the k -mer size plays an important role on the quality of the assembly and also creates trade-offs between several effects [40]. While short k -mers are expected to cover the original transcript fully and resolve the problems caused by errors in the sequences, they also create ambiguity because they can be shared among multiple transcripts. If repeats are longer than k , it creates forks in the graph, which causes the contig to break up. Longer k -mers, on the other hand, are expected to have higher chances of containing sequence errors. The errors in the k -mers cause the loss of overlap information, which affects the accuracy of the de Bruijn graph construction. In reality, it is difficult to determine which k -mer size generates the optimal assembly for a given data using a given assembler. Different assemblers result in different sets of transcripts even if they are used with the same k -mer size. When the same method is used with different k -mer sizes, assembly outputs can be also different.

Trinity includes three modules: Inchworm, Chrysalis, and Butterfly [52, 58]. Inchworm removes the erroneous k -mers from the read sequences, and then uses a greedy-

extension based overlap method to assemble reads into contigs. Chrysalis clusters the contigs and constructs a de Bruijn graph for each cluster. Finally, Butterfly traverses the graphs to construct transcripts. SOAPdenovo-Trans is an extension of the SOAPdenovo2 genome assembler [149, 88]. It uses the error removal methods of Trinity to remove edges representing the erroneous k -mers. The contigs extracted from the de Bruijn graphs are mapped to reads to build linkage between them, and the contigs are clustered into subgraphs based on the linkage information. Finally each subgraph is traversed to generate the transcripts. The default k -mer sizes for Trinity and SOAPdenovo-Trans are 23 and 25, respectively.

IDBA-Tran uses a unique assembly strategy [29]. It iterates k -mers from small to large k ($k = 20$ to 60 in every 10 in default) to balance the advantages and limitations of k -mer sizes. For each k -mer, it constructs a de Bruijn graph and then traverses the graph to generate contigs. The results from different k -mer sizes are merged by including the contigs generated with smaller k -mers as part of the input in the next iteration with a larger k -mer. rnaSPAdes is an extension of the SPAdes genome assembler [149, 88]. The de Bruijn graph used in SPAdes was modified for transcriptome assembly to handle paired-end reads, uneven coverage, and multiple insert sizes. Similar to IDBA-Tran, iterative de Bruijn graph construction was used but with only two k -mer sizes (one small and one large) dynamically selected using the input read data information.

6.2.3 Ensemble approach

No single assembler is considered to be the optimal for a wide range of input data [127, 143]. While it is possible to increase the true transcript reconstruction by combining the assembly results of multiple assemblers, this approach can also increase the number of mis-assembled transcripts. The ensemble approach of transcriptome as-

sembly attempts to reduce the number of mis-assembled transcripts without removing many correctly assembled transcripts. EvidentialGene [48] and the method proposed in [26] (we call this method “Concatenation”) merge multiple *de novo* assemblies and cluster contigs using either CD-HIT [80] or BLAST [4, 3] and select the representative sequences for the final assembly set. We previously reported a consensus strategy where multiple k -mers are considered for assembly and simple voting is used to select the contigs that are assembled by at least three out of four *de novo* assemblers for the final assembly set [143]. TransBorrow [151] is an ensemble approach that combines the results from different genome-guided assemblers. TransBorrow first extracts reliable subpaths supported by paired-end reads from a splice graph. Transcripts assembled by multiple genome-guided methods are merged and colored graphs representing the merged transcripts are built. Reliable assembly subpaths are further extracted based on the number of assemblers that detected each subpath (transcript). After combining reliable assembly subpaths and reliable subpaths on the splicing graphs, the final transcripts are assembled.

6.3 Performance Evaluation of Transcriptome Assembly

In order to evaluate the transcriptome assembly performance, we need to quantify the accuracy of assembled transcriptomes. Assembly performance metrics can be grouped into two classes: reference-free and reference-based. The reference-based metrics are further grouped into those based on real biological data and those based on simulated benchmark data.

6.3.1 Performance metrics without references

When high-quality reference sequences are not available to provide the ground truth, some assembly statistics can be used as reference-free performance. Some commonly used assembly statistics include:

- Number of contigs
- Median contig length (bp)
- N50 (or Nx): a length-weighted median where the sum of the lengths (bp) of all contigs longer than the N50 (or Nx) is at least 50% (or x%) of the total length of the assembly

rnaQUAST [23], for example, can be used to obtain these metrics. Higher values of N50 (Nx) indicate that a greater number of reads are overlapped to form longer contigs. In contrast to genome assembly, where longer contigs (e.g., larger N50) indicate a higher quality assembly, a transcriptome includes transcripts with varied lengths. The longer contigs in a transcriptome assembly could also represent over-assembly or chimeric contigs. Therefore, for a transcriptome assembly, the length-based metrics are not always useful as accuracy measures [102].

DETONATE provides a model-based score, RSEM-EVAL [78]. It combines the compactness of an assembly and the support of the assembly from the RNA-seq reads into a single score based on their joint probability. Higher RSEM-EVAL scores indicate better assembly performance.

TransRate [127] provides an assembly score based on four contig scores:

- $s(C_{nuc})$: measures the extent to which the nucleotides in the mapped reads are the same as those in the assembled contig

- $s(C_{cov})$: measures the proportion of nucleotides in the contig that have zero coverage
- $s(C_{ord})$: measures the extent to which the order of the bases in contig are correct
- $s(C_{seg})$: measures the probability that the coverage depth of the transcript is univariate, which represents a single-transcript assembly, not a hybrid/chimeric assembly
- r (TransRate assembly score): the geometric mean of the four contig scores multiplied by the proportion of RNA-seq reads that provide positive supports for the assembly (those map to the assembly)

6.3.2 Performance metrics using actual biological data

When the references (either genome or transcriptome sequences) are available, reference-based metrics can be calculated. rnaQUAST [23], for example, provides the gene-level metrics (e.g., numbers of assembled genes/isoforms/exons and their lengths) as well as the alignment metrics (e.g., numbers of aligned, unaligned, or misassembled transcripts).

DETONATE provides a tool kit, REF-EVAL [78], which computes a number of reference-based scores including:

- Recall, Precision, and F1: calculated at contig or nucleotide-level (see the equations [3] - [5] below)
- KC (k -mer compression) score: measures the accuracy of the assembly based on the weighted k -mer recall and the compression ratio between the assembly and the RNA-seq data

The quality of the assembly can be also evaluated based on the proportion of the predicted gene or protein sequences matched with those in the database of known genes or proteins. BUSCO [125], for example, provides a quantitative assessment of the completeness of an assembly in terms of the expected content of the lineage-specific gene dataset. The Benchmarking Universal Single-Copy Orthologs (BUSCO) is extracted from OrthoDB [72]. Orthologous candidate genes are searched at the protein level in the assembly and the results are summarized into five categories: complete and single-copy, complete and duplicated, fragmented, and missing.

In the comprehensive study reported in [60], these metrics were used to compare ten *de novo* assemblers using nine actual RNA-seq datasets.

6.3.3 Performance metrics using simulated benchmark data

Simulation can provide a way to generate benchmark datasets where the ground truth is known. This is the advantage over using the actual biological data as the reference, where the ground truth cannot be known completely. For a transcriptome analysis, RNA-seq can be simulated to generate short reads derived from a set of transcripts whose sequences are known. The simulated reads are used with assembly methods and the assembled contigs are compared with the original transcripts. This is also the only way where the information about the transcripts that are not assembled (missing transcripts) can be fully evaluated.

A contig generated by an assembler is considered to be correctly assembled (positive) if the identical sequence is present in the reference transcriptome in the benchmark dataset. A contig is considered to be mis-assembled (negative) if the identical sequence is not present in the reference transcriptome in the benchmark dataset. Note that less stringent performance evaluation can be performed by using a lower threshold ($< 100\%$) to identify positive contigs. It is also possible to use a protein-level

similarity instead of a nucleotide-level similarity to identify positive contigs. The test results are categorized as the following three outcomes:

- True positive (TP): a correctly assembled contig
- False positive (FP): a mis-assembled contig (including both partially correctly assembled and those with no similarity with the reference)
- False negative (FN): a benchmark transcript that is missing in the assembly

Note that true negative (TN) can be counted only if the benchmark dataset includes a negative transcript set (transcript sequences that do not belong to the reference set) and the assembly experiments are done including reads that are derived from negative transcripts.

The performance of each assembler is evaluated by the following metrics:

- Correct/incorrect ratio (C/I) = $\frac{TP}{FP}$ [1]

- Accuracy = $\frac{(TP+TN)}{(TP+FP+FN+TN)}$ or Accuracy* = $\frac{TP}{(TP+FP+FN)}$ [2]

- Recall (or Sensitivity) = $\frac{TP}{(TP+FN)}$ [3]

- Precision = $\frac{TP}{(TP+FP)}$ = 1 - False Discovery Rate (FDR) [4]

- F-measure (F or F1) = $\frac{(2(TP))}{(2(TP)+FP+FN)}$ [5]

In the equations above, TP , FP , TN , and FN are the numbers of instances in those categories. As shown in the equation [2], when TN is not counted, Accuracy cannot be calculated. In such cases, we define a modified accuracy (Accuracy*) without using TN .

The higher C/I shows that among the assembled contigs (predicted positives) there are more correctly assembled contigs (TP) than the mis-assembled contigs

(*FP*). This is similar to Precision where the proportion of correctly assembled contigs (*TP*) is shown relative to all assembled contigs. Recall also shows the proportion of correctly assembled contigs (*TP*) but relative to the number of transcripts in the reference (actual positives). Accuracy (or Accuracy*) and F-measure are combined metrics. F-measure is useful because it balances the concerns of Recall and Precision and does not require *TN* to be counted.

All the above metrics can be calculated at both the nucleotide and protein sequence levels. Depending on the transcriptome assembly algorithms, the 5' and 3'-ends of contigs are defined differently. Such small differences at the 5' and 3'-ends could have significant effects on the TP counts. By using the protein-level accuracy, this issue can be avoided. However, the performance metrics can also be affected depending on how the gene-prediction algorithm used to identify the open reading frame (ORF) from each contig works.

Although the assembly performance metrics calculated using simulated benchmark datasets are expected to provide better evaluation of the performance of transcriptome assemblers, challenges remain on how biologically realistic the simulation of RNA-seq data can be. If the read distribution and sequencing errors, for example, are not modeled properly, assemblers may perform well on simulated data but poorly on real data or vice versa.

6.4 Simulated Benchmark Transcriptome Datasets Generation

To analyze the performance of transcriptome assemblies, each of the benchmark transcriptome datasets should include the annotated genome, the transcriptome from which simulated RNA-seq is performed, and the RNA-seq data. In this section, we

first briefly describe the methods that can be used to simulate RNA-seq. We then discuss protocols to generate simulated benchmark datasets.

6.4.1 RNA-seq simulation methods

There are several tools that can simulate RNA-seq with short-read sequencing using the Illumina platform and/or third-generation long-read sequencing using the PacBio SMRT and ONT MinION platforms [156]. Many short-read simulators developed for benchmarking transcript abundance and differential expression tools, such as RSEM [77], SimSeq [18], SPsimSeq [5], and seqgendiff [47], model the error distribution and changes in transcript expression found in real RNA-seq datasets. This modeling can include sequence specific bias, such as producing fewer GC-rich reads [87], as in an extension to Polyester [45]. Some short-read simulators, such as Flux Simulator [53], attempt to reconstruct each step of the library preparation and sequencing pipeline, mimicking the errors and biases introduced at each step. Long-read simulators, including PBSIM [103], LongISLND [75], Badread [147], and Trans-Nanosim [56], focus on identifying the statistical distribution of read lengths and errors within the reads, especially the prevalence of insertions or deletions, which are common in long reads but rare in short reads. Note that while Trans-Nanosim is the only long-read simulator specifically built for RNA-seq data, all of these simulators have been applied to introduce sequencing errors to model transcriptomic data.

6.4.2 Examples of RNA-seq simulation

To illustrate how the RNA-seq simulation is done, for this example, we used Flux Simulator [53]. To model a range of transcriptome complexity, six genomes from four plant species including both monocots (*Oryza sativa* and *Zea mays*) and dicots (*Glycine max* and *Arabidopsis thaliana*) were chosen. The reference genome each

simulation was based is listed in Table 6.1. Using these genome sequences and gene annotations provided in gff files, RNA-seq simulation was performed as follows:

1. The expression profile was generated by Flux Simulator using the reference genome. Flux Simulator in default assigns random expression levels to genes and transcripts.
2. Fragmentation of the expressed transcripts was done using a uniform random distribution. For this example, the lengths were set to 300bp +/- 150bp. The fragments ≥ 150 bp were retained.
3. For sequencing, the Illumina Hi-Seq sequencing profile, which models sequencing errors, insert size, and transcript coverage, was used to generate 76 bp paired-end reads. For each transcriptome, a total of ~ 495 million reads were generated with more than 50X coverage for most transcripts.
4. For the reference set of transcripts, those that are mapped with sequenced reads with no gap in the coverage were chosen.
5. ORFfinder [146] was used to identify the ORFs from each reference transcript, and the longest ORFs was chosen.
6. After removing the redundant sequences, the benchmark transcriptome was obtained at both nucleotide and protein levels.

The detailed protocol is described in [145].

Existence of isoforms in transcriptomes can impact the assembly performance. As shown in Table 6.1, a significant variation in the number of isoforms was incorporated among the six benchmark datasets. The *Z. mays* B73 dataset has the highest level of isoform complexity. It contains more than 35% of the genes with two or more

isoforms and maximum number of isoforms in a gene is 20. In contrast, the majority of the genes (93%) in the dataset based on another strain of maize, *Z. mays* Mo17, have only one isoform (no alternative splicing). The *A. thaliana* No0 dataset has no multiple-isoform genes as the No0 reference transcriptome does not include isoform information, and hence each gene is represented by a single transcript. Although these datasets may not represent the actual distribution of isoforms in these genomes, they are useful for testing the impact of isoforms in transcriptome assembly.

In addition to incorporating isoforms, simulated benchmark datasets can be generated incorporating different levels of ploidy. Such simulation protocols can be found in, e.g., ([144]).

6.5 Performance Comparison among Transcriptome Assemblers

In this section, we will demonstrate how the performance among transcriptome assemblers can be compared using the simulated benchmark datasets prepared in the previous section.

Before running transcriptome assemblers, the simulated reads need to be pre-processed. We used the following settings:

- Quality filtering using Erne-filter 2.0 [43] with minimum mean Phred quality 20, 'ultra-sensitive' flag, and paired-end mode
- Read normalization using Khmer [34] with k -mer size of 32, an expected coverage of 50X, and paired-end mode

We compared the transcriptome assembly performance among three genome-guided (Cufflinks, StringTie, and Scallop), four *de novo* (IDBA-Tran, SOAPdenovo-

Trans, Trinity, and rnaSPAdes), and three ensemble (EvidentialGene, Concatenation, and the consensus approach) assemblers. For this analysis, performance metrics were calculated at the level of protein sequences. The longest ORF was identified by ORFfinder from each contig, and the translated ORF sequences were compared against the translated benchmark transcriptome. A contig was considered correctly assembled only if its coded protein sequence was identical to one of the translated benchmark transcripts.

6.5.1 Genome-guided approach

We used HISAT2 for aligning simulated short reads to the reference genomes before using the three genome-guided assemblers. To examine the effect of the reference genome, for *A. thaliana* and *Z. mays*, in addition to aligning each read set against the reference genome from which the simulated RNA-seq was performed, it was also aligned against the genome of the different strain of the same species. These results are shown as “same reference” and “different reference” in Table 6.1, respectively. The simplest test is the one with the *A. thaliana* No0 dataset, which does not include multiple isoforms for any gene, assembled using the same No0 genome as the reference. Surprisingly, no genome-guided methods had higher than 65% accuracy, with more than 25% of assembled contigs to be incorrect ($C/I \leq 3$). With more realistic isoform complexity, no method achieved the accuracy better than 50%. With both of the *Z. mays* datasets, more than half of assembled contigs were incorrect ($C/I \leq 1$). When these genome-guided methods were used with different references, although they are still from the same species, assembly performance deteriorated significantly: $< 22\%$ for the *A. thaliana* datasets and $< 10\%$ for the *Z. mays* datasets. For both *Z. mays* datasets, only 1 in 6 contigs were found to be correctly assembled ($C/I \leq 0.2$). It is notable that both *Z. mays* datasets generated lower quality assemblies

Table 6.1: Comparison of transcriptome assembly performance among different methods.^a

	Genome-guided (same reference)			Genome-guided (different reference)			<i>De novo</i> ^b				
	Cufflinks	StringTie	Scallop	Cufflinks	StringTie	Scallop	IDBA- Tran	SOAPdenovo -Trans	Trinity	rnaSPAdes	
[<i>A. thaliana</i> No0 (CS6805): 18,875 (100, 1, 1)] ^c											
	Reference: Col0										
# contigs ^d	19,288	21,027	21,397	21,178	20,264	18,817	22,768	29,773	23,476	27,664	
Accuracy*	0.62	0.65	0.61	0.18	0.22	0.22	0.25	0.30	0.40	0.28	
C/I	3.07	2.92	2.45	0.39	0.54	0.56	0.58	0.60	1.06	0.57	
[<i>A. thaliana</i> Col0 (TAIR9): 15,508 (79.03, 1.29, 8)] ^c											
	Reference: No0										
# contigs ^d	15,768	16,908	18,055	17,441	16,470	17,179	20,449	21,371	19,409	31,494	
Accuracy*	0.38	0.44	0.46	0.14	0.17	0.19	0.20	0.25	0.36	0.19	
C/I	1.20	1.43	1.42	0.30	0.39	0.45	0.42	0.52	0.92	0.32	
[Soybean (GCF_000004515.4): 18,215 (93.75, 1.07, 7)] ^c											
# contigs ^d	18,823	20,887	19,355				33,243	52,700	24,346	23,686	
Accuracy*	0.48	0.46	0.48				0.13	0.08	0.25	0.24	
C/I	1.77	1.44	1.67				0.22	0.12	0.53	0.52	
[Rice (GCF_001433935): 11,294 (97.97, 1.02, 3)] ^c											
# contigs ^d	10,200	9,344	11,436				13,151	18,000	10,508	13,182	
Accuracy*	0.39	0.40	0.48				0.16	0.17	0.30	0.28	
C/I	1.42	1.74	1.80				0.36	0.30	0.93	0.69	
[<i>Z. mays</i> B73 (GCF_000005005): 17,108 (74.08, 1.5, 20)] ^c											
	Reference: Mo17										
# contigs ^d	14,512	15,585	16,592	17,347	20,887	19,119	24,603	27,403	22,327	23,764	
Accuracy*	0.26	0.32	0.26	0.08	0.09	0.10	0.11	0.08	0.17	0.11	
C/I	0.79	1.04	0.71	0.18	0.18	0.21	0.20	0.13	0.35	0.20	
[<i>Z. mays</i> Mo17 (GCA_003185045.1): 17,479 (96.91, 1.04, 6)] ^c											
	Reference: B73										
# contigs ^d	18,163	24,388	21,572	18,543	21,944	19,257	24,916	26,257	21,537	21,469	
Accuracy*	0.29	0.24	0.26	0.08	0.08	0.08	0.13	0.09	0.18	0.16	
C/I	0.80	0.50	0.60	0.18	0.15	0.17	0.24	0.16	0.37	0.33	

^aThe best Accuracy* and C/I among all assemblers are shown in red. The scores in blue are the best among the *de novo* assemblers.

^bThe default kmer sizes were used for the *de novo* assemblers.

^cAfter each species name, the accession numbers of the reference genomic sequences used are shown in parentheses. The assembly of *A. thaliana* No0 (59) was downloaded from the 1001 genomes project (60). The assembly of *A. thaliana* Col0 was from the version 9 of the TAIR reference genome (61) and version 3 of the AtRTD transcriptome data set (62). The number after the colon is the total number of transcripts included in each benchmark dataset. The numbers in parentheses are % single-isoform gene, the average number of isoforms/gene, and the maximum number of isoforms/gene, in this order.

^dThe numbers of contigs are based on those unique at the protein sequence level.

compared to other datasets. A relatively lower quality of the *Z. mays* genomes may have contributed to the significantly poor performance of these assemblers with these datasets.

The overlap between correctly and incorrectly assembled contigs among the assemblies generated by the three genome-guided assemblers is illustrated in Figure 6.1. While each assembler generated a unique set of correct as well as incorrect contigs, $\sim 70\%$ or more of correctly assembled contigs were generated by all three assemblers. The exception was for the *Z. mays* B73 (37%) dataset. In contrast, the majority of incorrectly assembled contigs (62-87%) were uniquely generated by each assembler, and a very small number of contigs were incorrectly assembled by all three methods.

6.5.2 *De novo* approach

Each of the four *de novo* assemblers was run with the default parameters. As shown in Table 6.1, for all benchmark datasets, all *de novo* assemblers generated more contigs compared to genome-guided methods. However, their low accuracy (< 0.31) and C/I scores (< 0.63) indicate a large number of contigs were incorrectly assembled. Trinity, followed by rnaSPAdes, performed better than other *de novo* assemblers for all datasets. Interestingly while the *de novo* assemblers did not perform better than the genome-guided methods used with the same references, the performance of the *de novo* assemblers was better than the genome-guided methods when they were used with different references. Similar to the genome-guided assembly, the largest numbers ($\geq 30\%$ except 17% for the *Z. mays* B73 dataset) of the correctly assembled contigs were found in the group of contigs shared by all four *de novo* assemblers (Figure 6.2). Incorrectly assembled contigs were also found to be most likely assembled by individual assemblers uniquely and not shared with other assemblies.

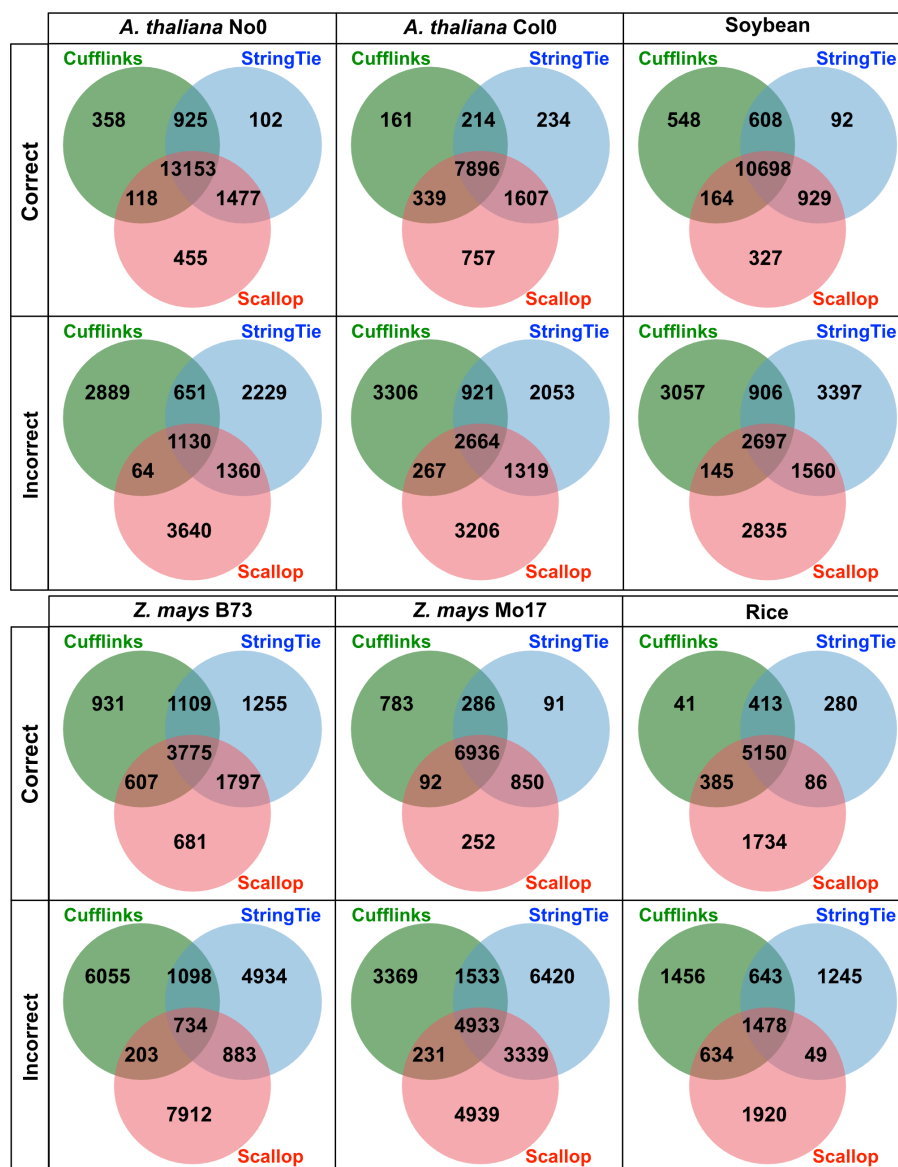


Figure 6.1: Numbers of correctly and incorrectly assembled contigs shared among the three genome-guided assemblers. Each genome-guided assembly was performed using the reference and the RNA-seq data from the same genome. Venn diagrams were generated using JVenn [9].

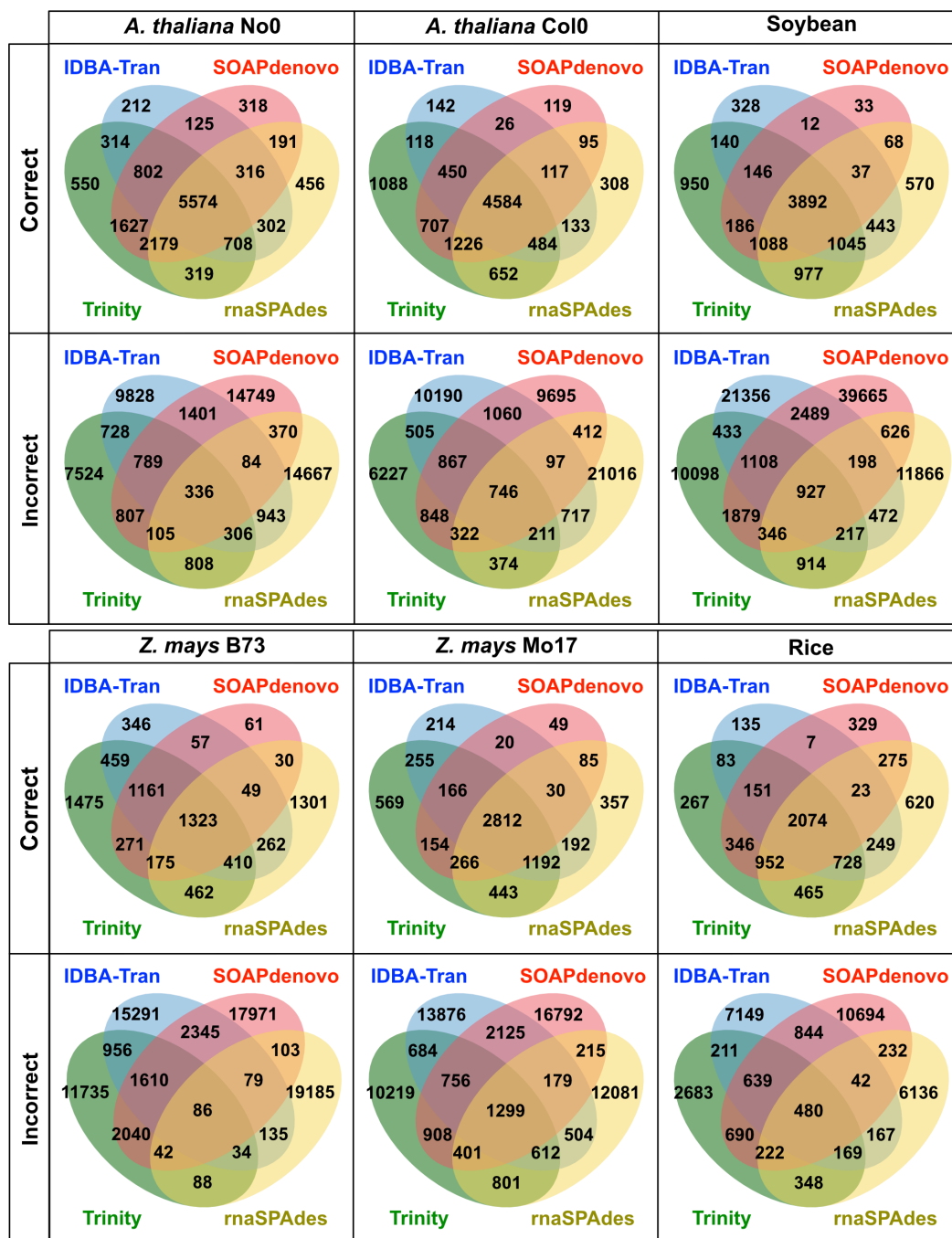


Figure 6.2: Numbers of correctly and incorrectly assembled contigs shared among the four *de novo* assemblers used with the default settings. Venn diagrams were generated using JVenn [9].

6.5.3 Combining *de novo* assemblies generated using different *k*-mers

Since the optimum *k*-mer size for each transcript assembly varies, different sets of correctly assembled contigs are expected even when the same *de novo* method is used with different *k*-mer sizes. Therefore, by combining the results from multiple *k*-mers, we expect to find more contigs correctly assembled by *de novo* assemblers. To illustrate this idea, for each of the four *de novo* assemblers, we used multiple *k*-mer sizes and generated a “pooled assembly” by combining their results (the union set). The four pooled assemblies are compared in Figure 6.3. Compared to Figure 6.2, the proportion of correctly assembled contigs shared by all four pooled assemblies increased significantly ($\geq 55\%$ except 36% for the *Z. mays* B73 dataset). Furthermore, only a very small proportion ($\leq 10\%$) of the incorrectly assembled contigs were shared by two or more pooled assemblies.

6.5.4 Analysis of *k*-mers used in assembled contigs

The *k*-mers of a contig that are not present in the benchmark transcriptome are considered to be false *k*-mers. When false *k*-mers are used for the de Bruijn graph construction in *de novo* assemblers, it generates incorrect contigs. To understand why the *Z. mays* B73 dataset generated poor assemblies regardless of the methods, we analyzed *k*-mers found in contigs assembled by the four *de novo* assemblers (Table 6.2). Compared to the assemblies generated from the Rice dataset, those generated from the *Z. mays* B73 dataset were represented by significantly lower numbers of true *k*-mers (the *k*-mers that are found in the benchmark transcriptome). Only for fewer than 50% of contigs assembled for the *Z. mays* B73 dataset, 90% or more of *k*-mers found were true *k*-mers. It appears that a large number of false *k*-mers were included in the de Bruijn graph construction for the maize transcriptomes leading to the poor

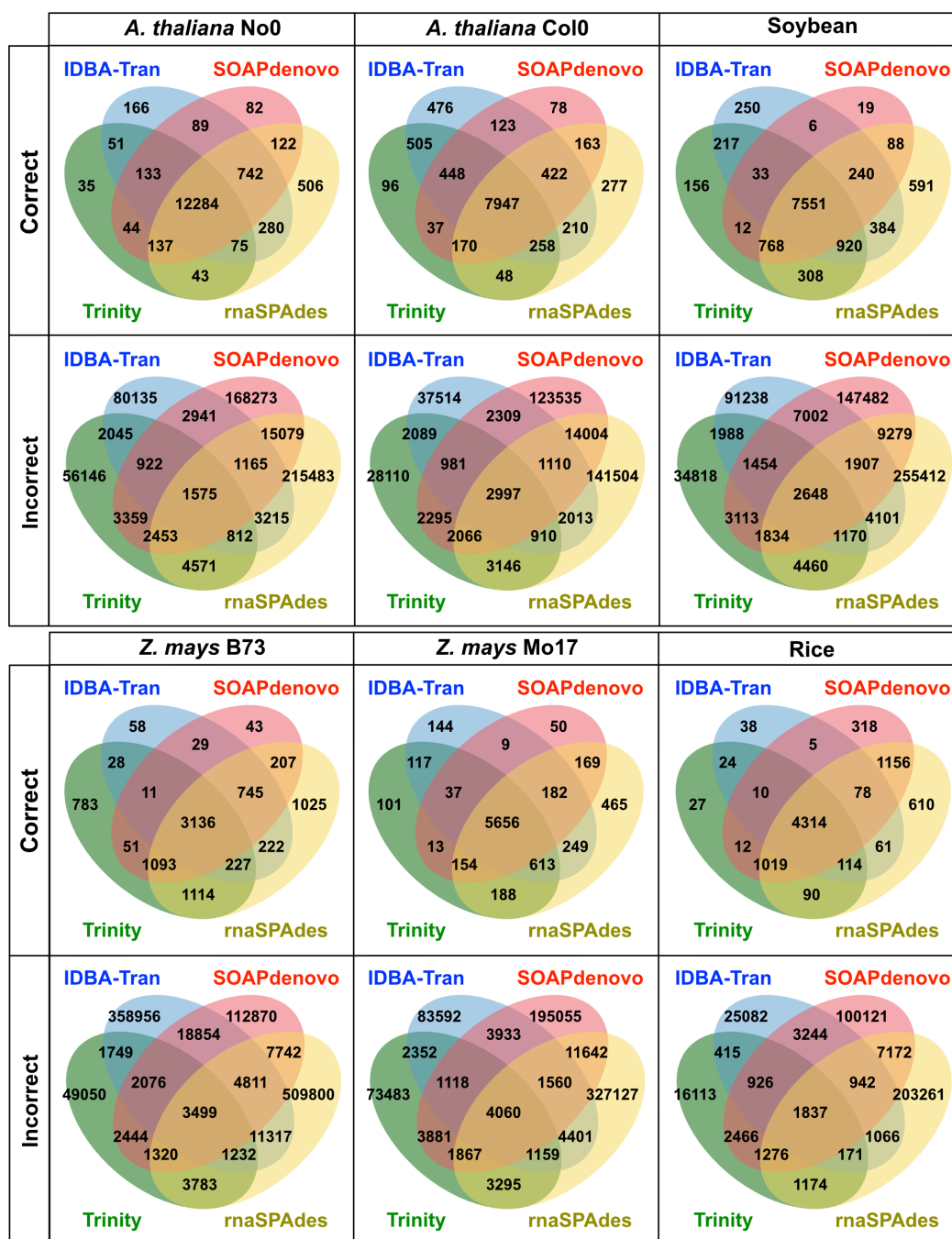


Figure 6.3: Numbers of correctly and incorrectly assembled contigs shared among the four pooled *de novo* assemblies. The following k -mers are used: for IDBA-Tran, $k=20$ 60 with increment of 10; for SOAPdenovo-Tran and rnaSPAdes, $k=19\sim 71$ with increment of 4; and for Trinity, $k=15\sim 31$ with increment of 4. Venn diagrams were generated using JVenn [9].

de novo assembly performance for this dataset.

Table 6.2: The k -mer analysis for the *de novo* assemblies using the *Z. mays* B73 and Rice datasets.^a

	IDBA-Tran	SOAPdenovo-Trans	Trinity	rnaSPAdes
[Rice]				
% true kmers ^b	96.09	97.56	98.89	55.27
% contigs with >90% true kmers ^c	95.96	92.86	97.68	58.91
[Z. mays B73]				
% true kmers ^b	26.18	48.7	53.57	27.72
% contigs with >90% true kmers ^c	15.23	47.6	38.7	21.81

^aAll results are based on pooled assembly.

^bThe proportion (%) of the kmers ($k=31$) found in the contigs that were also found in the benchmark transcripts (true kmers).

^cThe proportion (%) of the contigs where 90% or more of the kmer found were true kmers.

6.5.5 Ensemble approach

We finally compared the assembly performance of all individual methods with the three ensemble approaches (EvidentialGene, Concatenation, and the aforementioned consensus approach; see ([145]) for how these ensemble methods were used). Both EvidentialGene and Concatenation over-assembled and accumulated incorrectly assembled contigs as shown in their significantly higher Recall compared to Precision (Figure 6.4). It indicates that these methods recover many transcripts correctly at the expense of having a disproportionately large number of incorrectly assembled contigs. The F-measure (the combined score of Recall and Precision) scored lower for EvidentialGene and Concatenation compared to individual *de novo* assemblies for most of the datasets. It should be noted, however, that although many contigs retained by

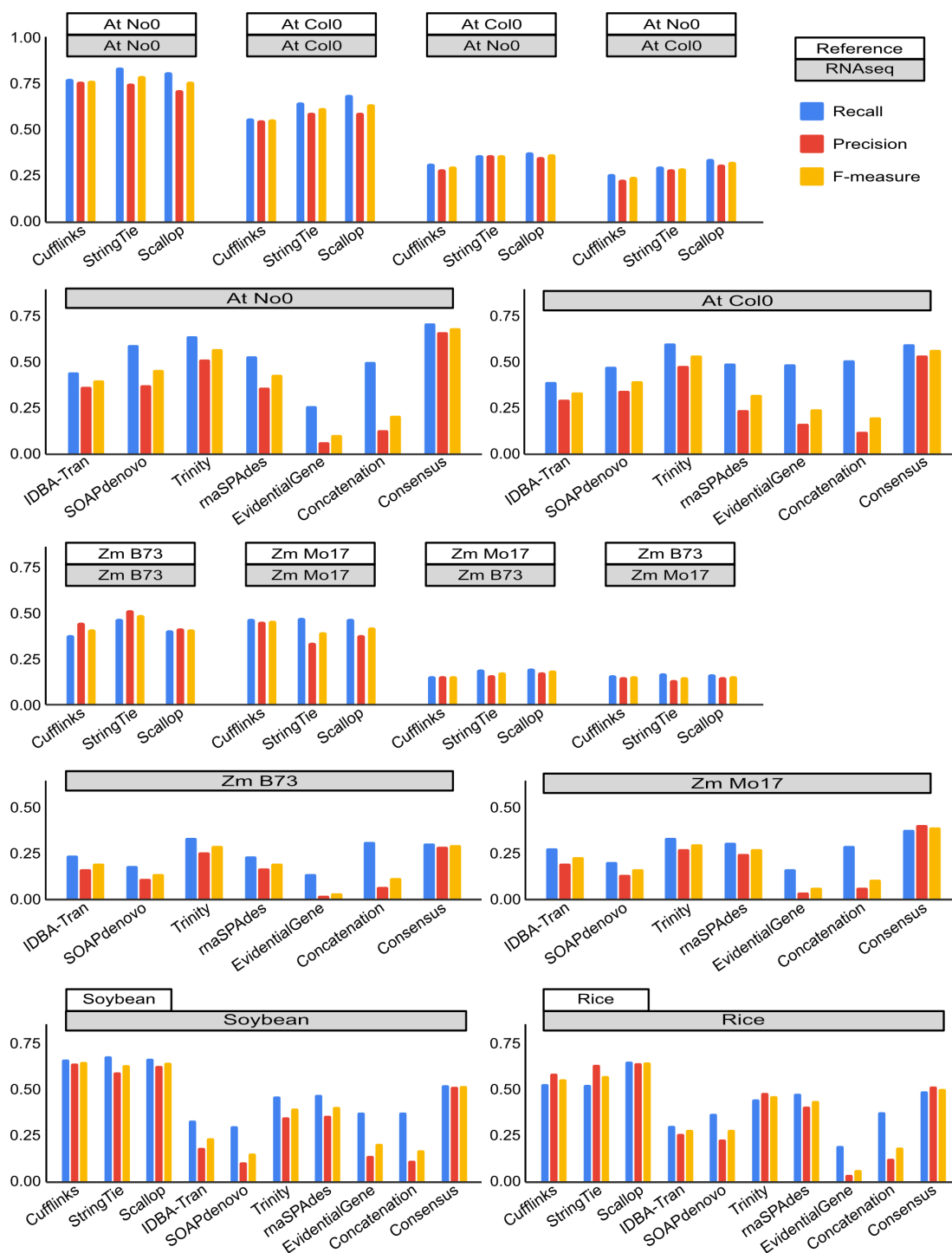


Figure 6.4: **Comparison of transcriptome assembly performance among different methods.** The simulated RNA-seq data (gray boxes) and the reference genome (for genome-guided methods; white boxes) used are shown at the top of each bar chart. The default k -mers were used for the *de novo* methods. At: *A. thaliana*, Zm: *Z. mays*.

these ensemble methods are identified to be incorrect, they are reported to be still highly similar ($> 98\%$) to the benchmark transcripts [145]. The consensus approach consistently performed better than all of the *de novo* assemblers for all datasets and achieved the performance similar to the genome-guided assemblers without requiring good reference genomes.

6.6 Minsemble: a New Ensemble Approach

Similar to other ensemble approaches such as EvidentialGene, Concatenation, and our recently developed consensus-based method, ConSemble [145], our new approach, Minsemble, makes use of assembly results from multiple *de novo* assemblers to improve accuracy of transcriptome assembly. Minsemble uses both clustering approach, as used in EvidentialGene and Concatenation, and the voting-based contig selection as used in ConSemble. However, the clustering process of Minsemble is isoform-based, i.e., grouping is based on the potential isoforms originated from the same gene. While the clustering of contigs for EvidentialGene, Concatenation, and ConSemble is used to remove redundancy, the goal of the isoform-based contig clustering used in Minsemble is to retain the isoforms for the final assembly. Minsemble follows three main steps.

- Minhash signature generation for each assembly
- Clustering of potential isoforms
- Selection of contigs for the final assembly

The main steps of the entire Minsemble method is shown in Figure 6.5 and the Minsemble pipeline for transcriptome assembly is shown in Figure 6.6.

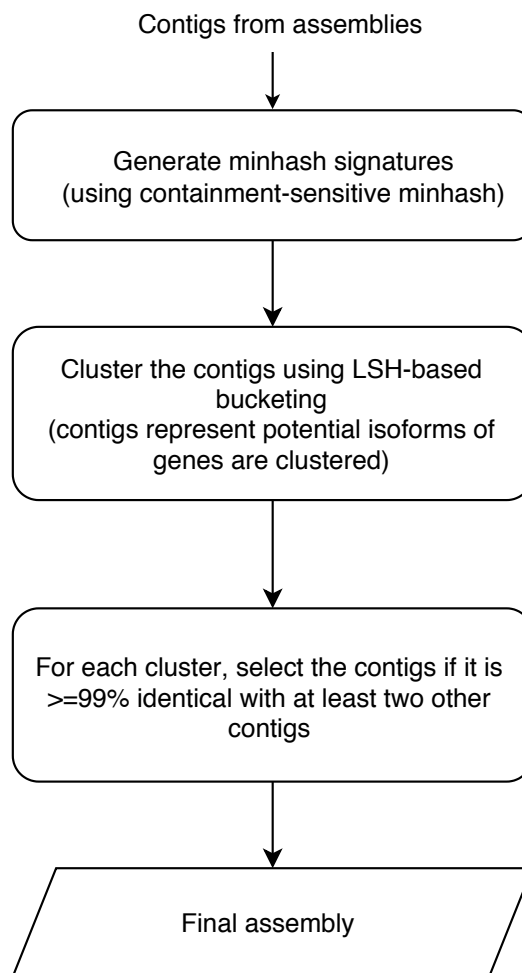


Figure 6.5: Minsemble procedure. See Figure 6.6 for the entire Minsemble transcriptome assembly pipeline.

6.6.1 Minhash signature generation

Computation of sequence similarity is the key to the clustering of sequences. The sequence similarity is usually calculated by using sequence alignment algorithms. For a large-scale analysis, it can be approximated by comparing q -gram sets. As shown in Figure 6.7, the sequences can be converted into sets of q -grams. The proportion of q -grams shared by sequences provides a good estimate of their similarity. The Jaccard similarity is often used to compute the similarity between two sets, which

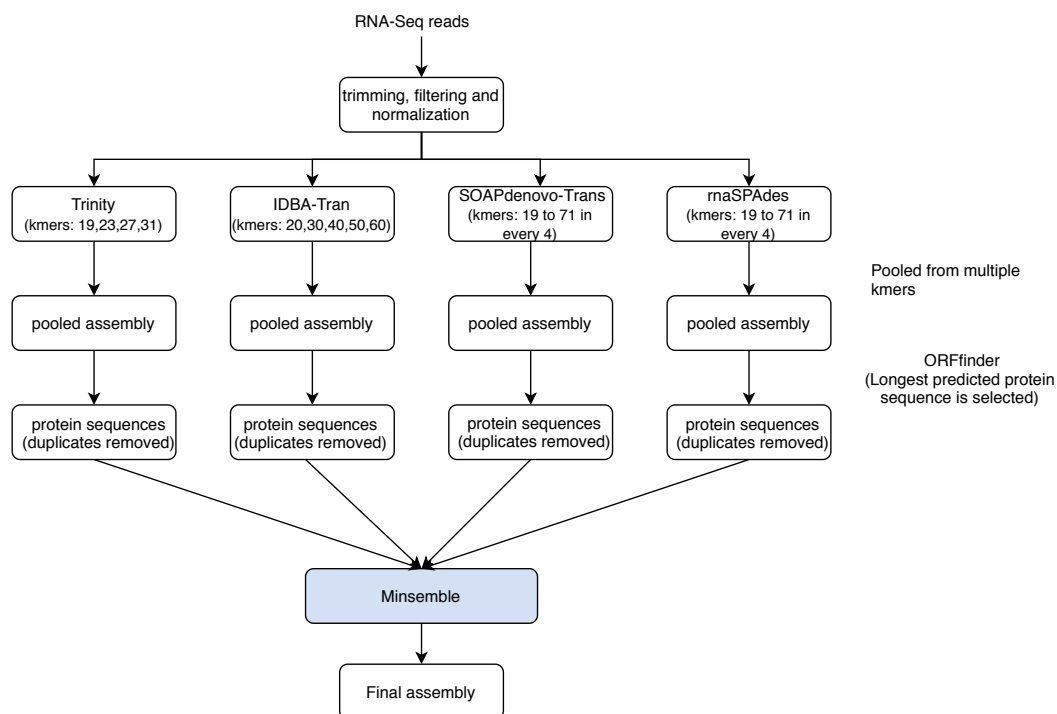


Figure 6.6: The Minsemble pipeline. The details for the Minsemble step are shown in Figure 6.5.

is calculated as the ratio of intersection over union. For sequence analysis, it can be achieved by decomposing sequences into q -gram sets and calculating the Jaccard similarity score between the two q -gram sets.

Minhash [22] is a technique to estimate the Jaccard similarity very efficiently for large datasets. It estimates by converting a set into a fixed-size integer signatures where the size of the signature is much smaller than the size of the sets. This technique has been used in many bioinformatics applications such as genome and metagenome distance estimation [101]. A minhash signature of a sequence is generated as follows. Let $Q = \{q_1, q_2, \dots, q_n\}$ be the set of all possible q -grams of a sequence of S , where n is the number of q -grams, and $H = \{H_1, H_2, \dots, H_N\}$ be the set of N hash functions. The hash functions are used to generate hash values for each q -gram of S . Using hash function H_i , let the set of hash values of all q -grams be $\{h_{1i}, h_{2i}, \dots, h_{ni}\}$ and

A. S_1 : ca~~a~~gtctagttata~~c~~gact-
 S_2 : ca-gtctagttata~~t~~gact~~t~~

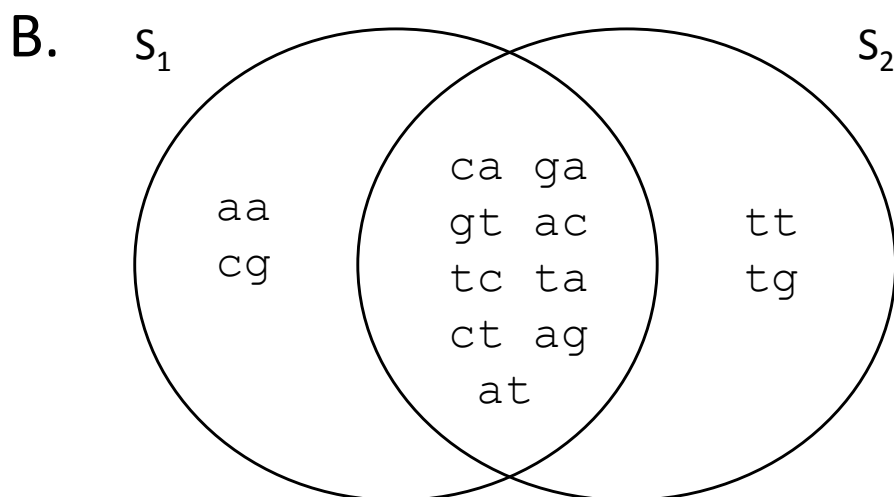


Figure 6.7: **Sequence similarity estimation using q -grams.** (A) The sequence alignment of sequences S_1 and S_2 . There are one mismatch (shown in blue) and two gaps (shown in red). These two sequences are 84% identical. (B) A Venn diagram of the two q -gram sets. In this example, the sequences are converted into a set of 2-grams. The Jaccard similarity score can be calculated as $J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{9}{13} \approx 0.7$.

the minimum value in this set be v_i . For each hash function in the set H , a set of hash values is generated and the minimum value among them is chosen. For N hash functions, $\{v_1, v_2, \dots, v_N\}$ becomes the set of minimum values. The set of minimum values generated with N hash functions is the minhash signature of the sequence S , $Sig(S)$. The main idea of minhash signature is to randomly pick a fixed number of q -grams from a set. Given two sequences S_1 and S_2 , the intersection-over-union score of the minhash signature sets of the q -gram sets of these two sequences provides a good estimate of the Jaccard similarity score, $J(S_1, S_2)$. The probability of two minhash

signatures being the same is a good approximation for the Jaccard similarity score between two sequences, i.e., $J(S_1, S_2) \approx Pr[Sig(S_1) = Sig(S_2)]$. The approximation becomes better when more hash functions are used for signature generation.

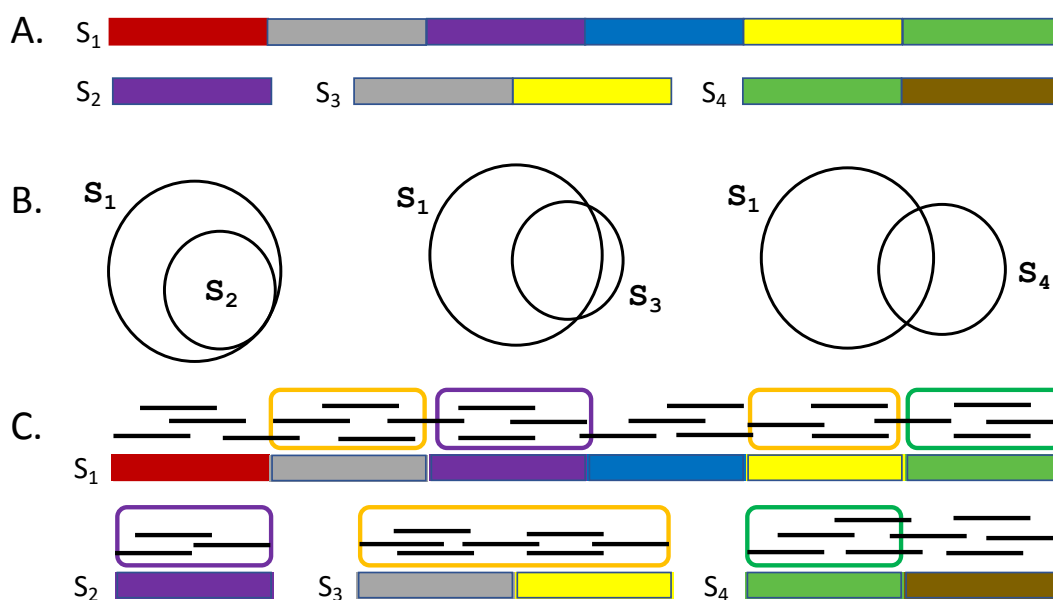


Figure 6.8: **Isoform and q -gram distribution.** (A) Isoform structures from a gene. Exon structures of four isoforms (S_1 - S_4) are shown. The same colors indicate the shared exons. For example, the gray exon is shared between isoforms S_1 and S_3 . (B) Distribution of q -grams among isoforms. q -grams from S_2 are completely contained in the q -gram set from S_1 . All exons of S_3 are also shared with S_1 . However, some q -grams derived from the junction region between the gray and yellow exons may not exist in S_1 because these two exons are not adjacent to each other in S_1 . S_4 has the brown exon that does not exist in S_1 . Therefore, S_4 has more unique q -grams compared to S_1 . (C) Fixed-size subsequences generated from each isoform for the containment-sensitive minhash. Subsequence regions shared between S_1 and other isoforms are indicated with colored rounded rectangles. q -grams are generated from each subsequence.

The minhash approach works well for the sequences of similar lengths. However, it does not work very well when the lengths of sequences differ significantly. For example, a shorter sequence can be completely contained in a longer sequence. In such cases, there is a high possibility that the q -grams of the longer sequence can be

picked from the segment that is not shared with the shorter sequence. As shown in Figure 6.8, isoforms share some but not all exons. However, some q -grams of a longer isoform, i.e., S_1 in Figure 6.8A, could come from the region that are not shared by other isoforms as shown in Figure 6.8B.

To address this problem, a containment-sensitive minhash approach [157, 70] can be used. This allows the clustering of potential isoform sequences from the same gene. In Minsemble, the containment-sensitive minhash approach is done as follows. Let l_{min} be the length of the shortest sequence in the set of input sequences. Each sequence s_i with length l_i is divided into subsequences of length l_{min} at r random positions where $r = 3 \times \frac{l_i}{l_{min}}$ and $l_i > l_{min}$ (6.8C). The choice of r ensures that there are at least three overlapping subsequences for each position. While a higher r value increases the number of overlapping regions and is expected to make better approximation, our experiments on test data showed that $r = 3$ gives a satisfactory approximation and very few false negatives. For each subsequence, q -grams are extracted, the minhash signature is generated, and the final signature of a sequence is represented as a set of minhash signatures.

6.6.2 Clustering of potential isoforms

Estimating the Jaccard similarity using minhash signatures for each pair of sequences is not computationally efficient. The LSH-based bucketing method [22, 31] can be used to cluster the sequences efficiently by avoiding performing too many pairwise operations. LSH is a hashing technique to map two similar items into the same hash value. The probability of two sequences being similar depends on the number of shared elements in their minhash signatures. The LSH-based method makes use of information about shared elements in minhash signatures to create buckets and put similar sequences into the same bucket as follows. The minhash signatures are divided

into b bands with each band containing $\frac{N}{b}$ elements. A hash value that is generated for each band becomes the signature of the bucket. If two signatures share the same band, then they should share the same bucket. Bucket IDs are created for each band and the sequence is assigned to the bucket corresponding to the band it contains. If two sequences are similar or two segments of two sequences are similar, then there is a high possibility that their corresponding minhash signature shares a band. Thus the two sequences will be put into the same bucket. As a sequence can have b bands, at most b buckets can contain that sequence. For two similar sequences, it may happen that all b buckets contain the two sequences. Therefore, the LSH-bucketing step could produce many redundant buckets, i.e., buckets sharing the same set of sequences. It is also possible that one bucket may contain a sequence that is not similar to others but share a band with them by chance. Therefore, finding and removing potential false positives needs to be done by checking pairwise similarity within each bucket. For isoforms that share some exons but are different in the middle exons are included in the same bucket by using a bloom filter based q -gram membership test [15]. If two isoforms from same gene do not share any exons, then it is possible that those two isoforms are included in two different buckets.

6.6.3 Selection of contigs for final assembly

As described in section 6.5 ([145, 17]), using simulated benchmark datasets, we showed that the chances of a contig being correctly assembled increases with the increase of the number of assemblers that share that contig. As described in section 6.5.5 (Figure 6.4), our previously developed consensus-based approach, ConSemble, where the contigs that are identical in coded protein sequences among at least three assemblers are retained, showed significant improvement over other ensemble approaches such as EvidentialGene and Concatenation. However, this approach still missed more than

```

A. IDBA_contig1      MVGYNNKKCWPRDARMRLMKHDVNLGRSVFVWDMKNRLPRSITTTLEWENGFVSVYSKDNP
SOAP_contig1       MVGYNNKKCWPRDARMRLMKHDVNLGRSVFVWDMKNRLPRSITTTLEWENGFVSVYSKDNP
SPAdes_contig1    MVGYNNKKCWPRDARMRLMKHDVNLGRSVFVWDMKNRLPRSITTTLEWENGFVSVYSKDNP
Trinity_contig1   MVGYNNKKCWPRDARMRLMKHDVNLGRSVFVWDMKNRLPRSITTTLEWENGFVSVYSKDNP

IDBA_contig1      LLFSMCGFEVRILPKIRMTQEAFSNTKDGWVNQMLLSDRFLGFYMPVPSGLQNEQTKERT
SOAP_contig1     LLFSMCGFEVRILPKIRMTQEAFSNTKDGWVNQMLLSDRFLGFYMPVPSGLQNEQTKER--
SPAdes_contig1   LLFSMCGFEVRILPKIRMTQEAFSNTKDGWVNQMLLSDRFLGFYMPVPSGLQNEQ-----
Trinity_contig1  LLFSMCGFEVRILPKIRMTQEAFSNTKDGWVNQMLLSDRFLGFYMPVPSG-----

B. IDBA_contig2      MPMPEQVRDVKVLYHITGAIITFVNEIPWVVEPIYMAQWGTWIMMRREKRDRRHFKRMR
SOAP_contig2     MPMPEQVRDVKVLYHITGAIITFVNEIPWVVEPIYMAQWGTWIMMRREKRDRRHFKRMR
SPAdes_contig2   QPMPEQVRDVKVLYHITGAIITFVNEIPWVVEPIYMAQWGTWIMMRREKRDRRHFKRMR
Trinity_contig2  MPMPEQVRDVKVLYHITGAIITFVNEIPWVVEPIYMAQWGTWIMMRREKRDRRHFKRMR

IDBA_contig2      FPPFDDEEPPLDYADNLLDVPLEPIQLELDEEEDSAVHTWFDHKLKLVKTKLINGPSYR
SOAP_contig2     FPPFDDEEPPLDYADNLLDVPLEPIQLELDEEEDSAVHTWFDHKLKLVKTKLINGPSYG
SPAdes_contig2   FPPFDDEEPPLDYADNLLDVPLEPIQLELDEEEDSAVHTWFDHKLKLVKTKLINGPSTR
Trinity_contig2  FPPFDDEEPPLDYADNLLDVPLEPIQLELDEEEDSAVHTWFDHKLKLVKTKLINGPSPD

C. IDBA_contig3      PQLSPQDVTSHSRILENNKQWDGEKCIILTCSFTPGSCSLTSYKLTQTGYEWGRLNKDNP
SOAP_contig3     -----SRILENNKQWDGEKCIILTCSFTPGSCSLTSYKLTQTGYEWGRLNKDNP
SPAdes_contig3   -----PQDVTSHSRILENNKQWDGEKCIILTCSFTPGSCSLTSYKLTQTGYEWGRLNKDNP
Trinity_contig3  -----PIILTCSFTPG-CSLTSYKEWETGFVWGRLNDNPN

IDBA_contig3      SNPHGYLPTHYEKVQM-LSDRFLGFYMPVPSGPWNYSFTGVKHTLSMKYSVKLGSPKEFF
SOAP_contig3     SNPHGYLPTHYEKVQM-LSDRFLGFYMPVPSGPWNYSFTGVKHTLSMKYSVKLGSPKEGF
SPAdes_contig3   SNPHGYLPTHYEKVQM-LSDRFLGFYMPVPSGPWNYSFTGVKHTLSMKYSVKLGSPKEWP
Trinity_contig3  LLFSMCGFEVRILPKIRMTQEAFSNTKDGWVNQMLLSDRFLGFYMPVPSG-----

```

Figure 6.9: **Retention of highly similar contigs and potential isoforms.** Contigs (in coded protein sequences) in the same cluster are aligned. Positions with mismatches or gaps are shown in red color. (A) All contigs will be retained for the final assembly as pairwise similarity is all 100% excluding the gaps at the end regions. (B) All contigs will be retained for the final assembly as pairwise similarity is all $> 99\%$. (C) Only first three contigs are retained as their pairwise similarity is $\geq 99\%$. The last contig is $\sim 83\%$ identical to the first three, and hence will not be retained.

40% of true contigs for the datasets that included isoforms and more than 30% for the datasets that did not contain isoform (Figure 6.4). Furthermore, for genes that have multiple isoforms, the majority of genes were represented by only one isoform and other isoform sequences were omitted from the final assembly [145]. Our performance analysis based on the benchmark datasets showed that different assemblers correctly assembled different sets of transcripts (Figure 6.2). Therefore, it is possible that the entire sequences of these contigs are not shared by three or more assemblers. Furthermore, we observed that many assembled contigs generated by some assemblers were not 100% identical to the benchmark transcript but include a small number of

mismatches.

Our method, Minsemble, addresses these challenges by reducing the sequence identity threshold from 100% as used in ConSemble. In Minsemble, contig sequences in each clusters generated in the previous step are examined, and the contigs that are at least 99% identical at the protein level to at least two contigs generated by two other assemblers. In each cluster, pairwise edit distances are calculated for each contig against all other contigs using edlib [159] with the semi-global alignment (HW or infix) mode where the start- and end-gaps are not penalized. In Figure 6.9, some of the examples are shown to illustrate how the contig retention is done. In Figure 6.9A and B, all four contigs are retained as all contigs are 99% or more identical to other contigs. The last contig in Figure 6.9C has many mismatches when compared with other three (similarity score $\sim 83\%$). Therefore, it is not retained.

6.6.4 Minsemble transcriptome assembly pipeline

Following our previous ensemble transcriptome assembler, ConSemble, Minsemble makes use of contigs generated from multiple assemblers with multiple k -mer values. The entire Minsemble pipeline is illustrated in Figure 6.6. After quality filtering and normalization (see Section 6.5), four *de novo* assemblers (Trinity, IDBA-Tran, SOAPdenovo-Trans, and rnaSPAdes) are run with multiple k -mer values (see Figure 6.6 and [145] for the selection of k -mer values for each assembler). The assembled contigs from multiple k -mer values are pooled together to generate the “pooled assembly” for each assembler. ORFfinder [1] is used to predict the longest protein-coding region from each contig sequence. The Minsemble clustering as described before and summarized in Figure 6.5 is performed at the protein sequence level. The final assembly includes all retained protein and corresponding contig sequences with isoform-cluster information.

6.6.5 Assembly performance evaluation

The performance of each assembler was evaluated at both the transcript level and the gene level using six simulated benchmarked datasets as described in 6.4 and [17] as well as the Human dataset described in [145]. A contig generated by an assembler is considered to be correctly assembled if it matches 100% with one of the benchmark transcripts at the protein level. A contig is considered to be incorrectly assembled if no benchmark transcript is 100% identical to the contig sequence at the protein level. At the transcript level, true positives (TPs) and false positives (FPs) are simply correctly and incorrectly assembled contigs, respectively. False negatives (FNs) are the transcripts that are found in the benchmark transcriptome but are not correctly assembled.

At the gene level, the performance of assemblers were evaluated for the following three different groups: (1) for all genes, (2) for single-isoform genes, and (3) for multiple-isoform genes. For each gene included in the benchmark dataset, the isoform information is included based on the original genomic annotation. For each gene group, the number of genes whose transcripts are correctly identified was counted. For both of the “all gene” and “single-isoform gene” groups, a gene is defined as correctly identified if at least one transcript is assembled correctly. For the “multiple-isoform gene” group, a gene is defined as correctly identified if all transcripts (isoforms) are correctly assembled.

The contigs that are generated by the assemblers except Minsemble are not grouped based on the gene or isoform groups. To calculate detailed performance metrics at the gene level, assembled contigs also need to be clustered at the level equivalent to genes. We, therefore, used MinIsoClust [15] to cluster the contigs generated by other assemblers for potential isoforms for each gene where the number

of clusters is equivalent to the number of genes (Ng). By comparing the isoform groups in benchmark datasets and the isoform clusters generated by Minsemble or MinIsoClust (for other assemblers), TP, FP, and FN can be identified as follows.

For the “all gene” group, TPs are defined to be the genes in the benchmark dataset for which at least one isoform is correctly assembled by a given assembler. FNs are defined as the genes in the benchmark dataset for which no isoform is correctly assembled by a given assembler. FPs are the isoform clusters generated by Minsemble or MinIsoClust where none of the contigs match any of the benchmark transcripts. Therefore, $FP = Ng - TP$, where FP and TP are the numbers of FPs and TPs, respectively.

For the “single-isoform gene” group, only the genes in the benchmark dataset that have only one transcript are compared against the singleton clusters generated by Minsemble or MinIsoClust. TPs are the single-isoform genes in the benchmark dataset that are correctly assembled by a given assembler and present in singleton clusters. FNs are single-isoform genes in the benchmark dataset that are either not assembled by a given assembler correctly or not found in singleton clusters. FPs are singleton contigs in clusters generated by Minsemble or MinIsoClust that are not correctly assembled.

For the “multiple-isoform gene” group, the genes that contain multiple isoforms are compared against the contig clusters that also contain multiple isoform candidates. TPs are defined to be the genes in the benchmark dataset for which all isoforms are correctly assembled by a given assembler. FNs are the genes in the benchmark dataset for which not all isoforms are correctly assembled by a given assembler. FPs are defined to be the isoform clusters for which any of the followings are true: (a) no contigs are correctly assembled, (b) the cluster contains only some isoforms of a gene correctly assembled, or (c) the cluster contains correctly assembled isoforms of more than one

genes.

The metrics used in the performance analysis are described in 6.3.

6.6.6 Results and discussion

The transcriptome assembly performance by Minsemble was compared to other ensemble approaches (EvidentialGene, Concatenation, and ConSemble) as well as four individual *de novo* assemblers (IDBA-trans, Trinity, SOAPdenovo-Tran, and rnaSPAdes) using the seven simulated benchmark datasets. As described in 6.4.2, these datasets represent ranges of isoform complexity (see Table A.1 for isoform distribution in each dataset). The performance of all assemblers were evaluated at both the transcript level as well as the gene level.

6.6.6.1 Performance of transcriptome assembly at the transcript level

As Minsemble uses voting approach similar to ConSemble, it is expected that Minsemble will retain all TPs that are generated by ConSemble. With the slightly relaxed threshold (retaining threshold $\geq 99\%$), for the final assembly, Minsemble can retain more correctly assembled contigs, although it can also increase *FP*. The performance of Minsemble is compared with the four *de novo* and three ensemble methods in Figure 6.10 (also see Supplemental Tables A.2 - A.8). For all benchmark datasets, Minsemble showed the highest Recall values (0.74 for *A. thaliana* No0 and 0.66 for *A. thaliana* Col0, for example), followed usually by ConSemble and Trinity. The high Recall values indicate that Minsemble assembled more contigs correctly. However, ConSemble as well as Trinity and sometimes rnaSPAdes often showed higher Precision than Minsemble. This is expected because Minsemble retains many similar contigs ($\geq 99\%$ identity) including potential isoforms. Although these highly similar contigs are clustered together, these extra contigs are considered to be FPs

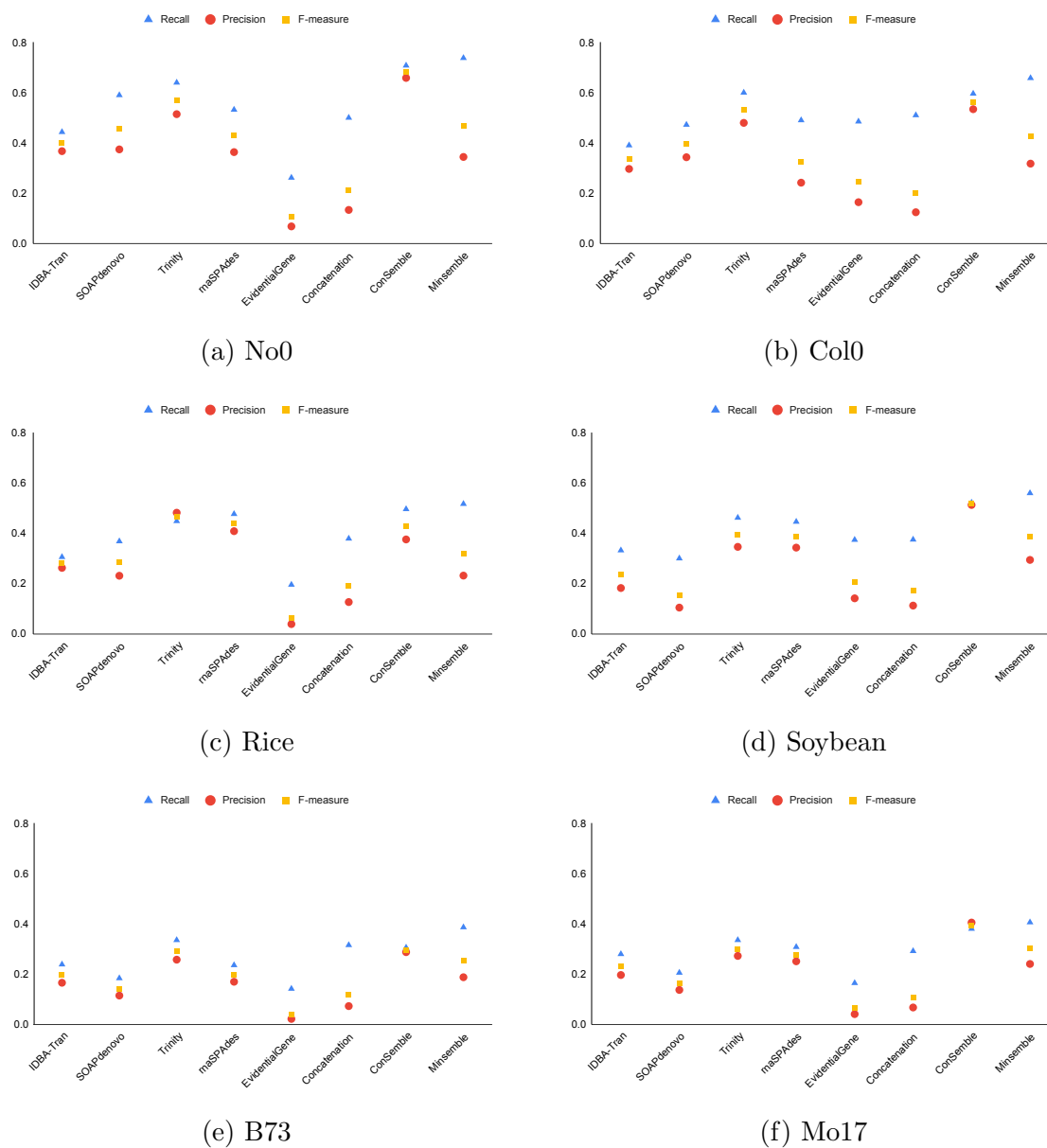


Figure 6.10: **Transcriptome assembler performance at the transcript level.** Performance was compared among four *de novo* and four ensemble assembly methods. The detailed performance metrics are given in Supplementary Tables A.2 - A.8.

at the strict threshold we used (100% identity). Note, however, that both Precision and F-measure values were much higher with Minsemble compared to the other two ensemble methods (EvidentialGene and Concatenation).

The *A. thaliana* No0 dataset contains only single-isoform genes (no alternatively spliced transcripts) and this is the simplest benchmark dataset. Therefore, as expected, all transcriptome assemblers performed the best for this dataset. The *Z. mays* B73 dataset, on the other hand, has the most complex isoform distribution (see Supplementary Table A.1). The assembler performance was, therefore, significantly worse for the *Z. mays* B73 dataset but it is also the case for the *Z. mays* Mo17 dataset where the isoform complexity is much less. The quality of the reference genome affects the performance of transcriptome assemblers as simulated benchmark sequences are generated using the reference genome. It is, therefore, possible that the erroneous annotation of *Z. mays* reference sequences affected the weak performance of all transcriptome assemblers.

Correctly and incorrectly assembled contigs were compared among the assemblies generated by four ensemble assemblers and their overlaps are illustrated in Figures 6.11 and 6.12 (see also Supplementary Figure A.1 for the Human dataset). As noted before, while the Minsemble assembly contains all contigs generated by ConSemble, Minsemble assembled more contigs correctly. The majority of correctly assembled contigs are shared among all four ensemble approaches except for the Rice and the *Z. mays* B73 datasets. Furthermore, for all datasets, the contigs shared among all four methods are more likely to be correct than incorrect ($C/I = 1.9 \sim 9.6$). In contrast, the contigs assembled uniquely by each ensemble method are highly likely to be incorrect ($C/I < 0.025$) regardless of the method.

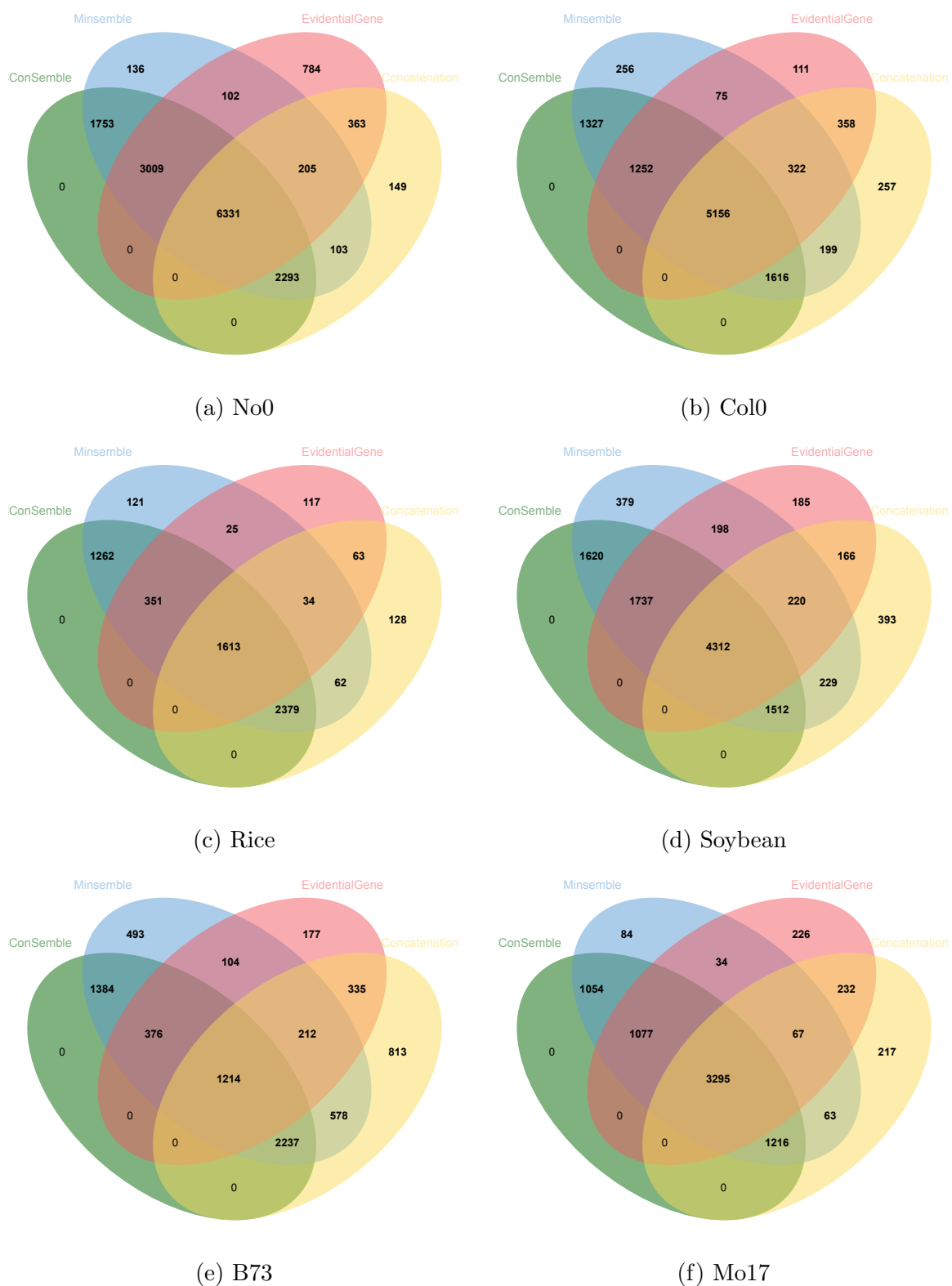


Figure 6.11: Numbers of correctly assembled contigs shared among the four ensemble assembly approaches.

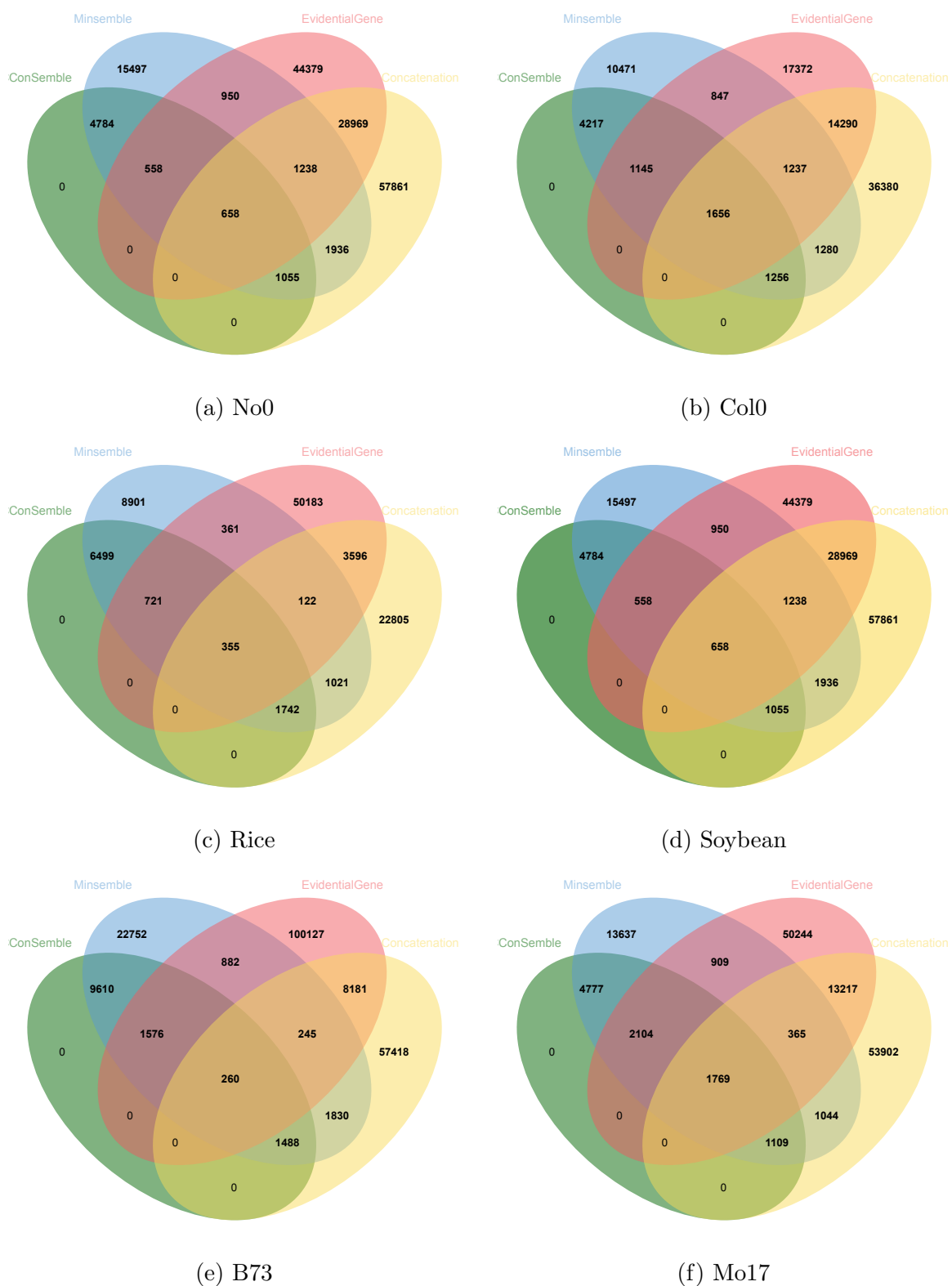


Figure 6.12: Numbers of incorrectly assembled contigs shared among the four ensemble assembly approaches.

6.6.6.2 Performance of transcriptome assembly at the gene level

In order to understand the advantage of Minsemble more clearly, we further examined the performance of the four *de novo* and four ensemble assemblers at the gene level. The gene identification performance of the four *de novo* assemblers and the four ensemble approaches was compared in Table 6.3. For all benchmark datasets, Minsemble showed the best performance in terms of gene identification.

In Tables 6.4 and 6.5, the gene identification performance was examined in two gene groups, “single-isoform genes” and “multiple-isoform genes”. Minsemble performed the best for identifying the single-isoform genes, closely followed by ConSemble. For identification of the “multiple-isoform genes”, Minsemble clearly performed better than all other methods. Note also that for the *Z. mays* B73 dataset, all assemblers identified only < 10% of the isoforms correctly. This indicates that, the performance of assemblers was affected by the complex isoform distribution. As compared to all assemblers, Minsemble identified more genes correctly especially for those with multiple isoforms, indicating the advantage of using Minsemble.

Table 6.3: Gene identification performance of transcriptome assemblers^a

Dataset	IDBA-Tran	Trinity	SOAPdenovo	rnaSPAdes	Concatenation	EvidentialGene	ConSemble	Minsemble
No0	56.42	61.56	44.84	58.38	51.39	53.29	65.22	66.94
Col0	50.24	58.28	49.80	61.13	56.78	56.51	69.99	71.89
Rice	10.93	11.85	28.67	24.59	38.43	19.86	46.70	49.12
Soybean	25.46	22.90	37.21	35.74	38.80	30.56	52.93	57.56
B73	16.67	12.17	7.81	20.39	40.16	20.00	44.10	51.92
Mo17	28.92	33.89	21.22	31.54	29.67	28.93	38.54	46.34
Human	64.59	64.59	48.82	58.31	45.90	47.74	67.49	70.45

^a% gene identification is based on “all genes” included in each benchmark dataset. A gene is defined as correctly identified if at least one transcript is assembled correctly. The highest values are shown in bold face.

With Minsemble, contigs are clustered based on possible isoform groups, which is equivalent to assigning contigs to each gene. However, for other *de novo* transcriptome assemblers, such is not possible. Without the potential genes identified in each assembly, it is not possible to calculate detailed performance metrics at the gene level as we did for the transcript level analysis. Therefore, for all assemblies

Table 6.4: Gene identification performance of transcriptome assemblers for the single-isoform genes^a

Dataset	IDBA-Tran	Trinity	SOAPdenovo	rnaSPAdes	Concatenation	EvidentialGene	ConSemble	Minsemble
No0	56.42	61.56	44.84	58.38	51.39	53.29	65.22	66.94
Col0	56.84	63.41	45.35	59.23	51.48	53.34	65.27	66.96
Rice	10.41	12.07	28.48	24.69	37.78	19.54	46.15	48.72
Soybean	19.18	14.09	33.73	29.94	36.38	28.99	50.27	54.48
B73	11.67	10.61	8.05	15.09	31.45	16.75	41.30	45.78
Mo17	28.76	33.01	20.90	30.85	28.80	28.12	37.44	44.92
Human	48.80	54.19	48.59	51.02	38.82	42.96	54.76	64.51

^a% gene identification is based on only single-isoform genes included in each benchmark dataset. A gene is defined as correctly identified if the single transcript is assembled correctly. The highest values are shown in bold face.

Table 6.5: Gene identification performance of transcriptome assemblers for multiple-isoform genes^a

Dataset	IDBA-Tran	Trinity	SOAPdenovo	rnaSPAdes	Concatenation	EvidentialGene	ConSemble	Minsemble
Col0	5.83	23.05	13.53	19.75	26.85	11.54	23.76	39.87
Rice	3.13	1.34	0.89	2.68	4.02	3.57	8.48	21.88
Soybean	16.56	14.26	18.40	15.49	18.10	11.35	25.00	55.06
B73	1.59	1.42	1.12	2.67	2.87	2.23	2.77	9.47
Mo17	2.50	22.65	4.03	10.36	13.05	11.13	23.99	48.56
Human	5.27	11.47	4.41	9.24	7.12	6.95	14.88	18.98

^a% gene identification is based on only the multiple-isoform genes included in each benchmark dataset. A gene is defined as correctly identified if all of the isoforms are assembled correctly. The highest values are shown in bold face.

other than those generated by Minsemble, we used MinIsoClust to identify potential isoform clusters from each assembly result. An isoform cluster was considered to be a gene and a gene was determined to be correctly identified if at least one isoform was correctly assembled at the protein level. Assembly performance was compared among the four *de novo* and the four ensemble methods in Figure 6.13 (also see Supplementary Tables A.9 - A.14). Minsemble and ConSemble both clearly performed better than other assemblers. Similar to the transcript-level performance, Minsemble always had higher Recall values compared to ConSemble. Furthermore, for the Rice, Soybean, and *Z. mays* Mo17 datasets, Minsemble performed better than ConSemble in terms of F-measure.

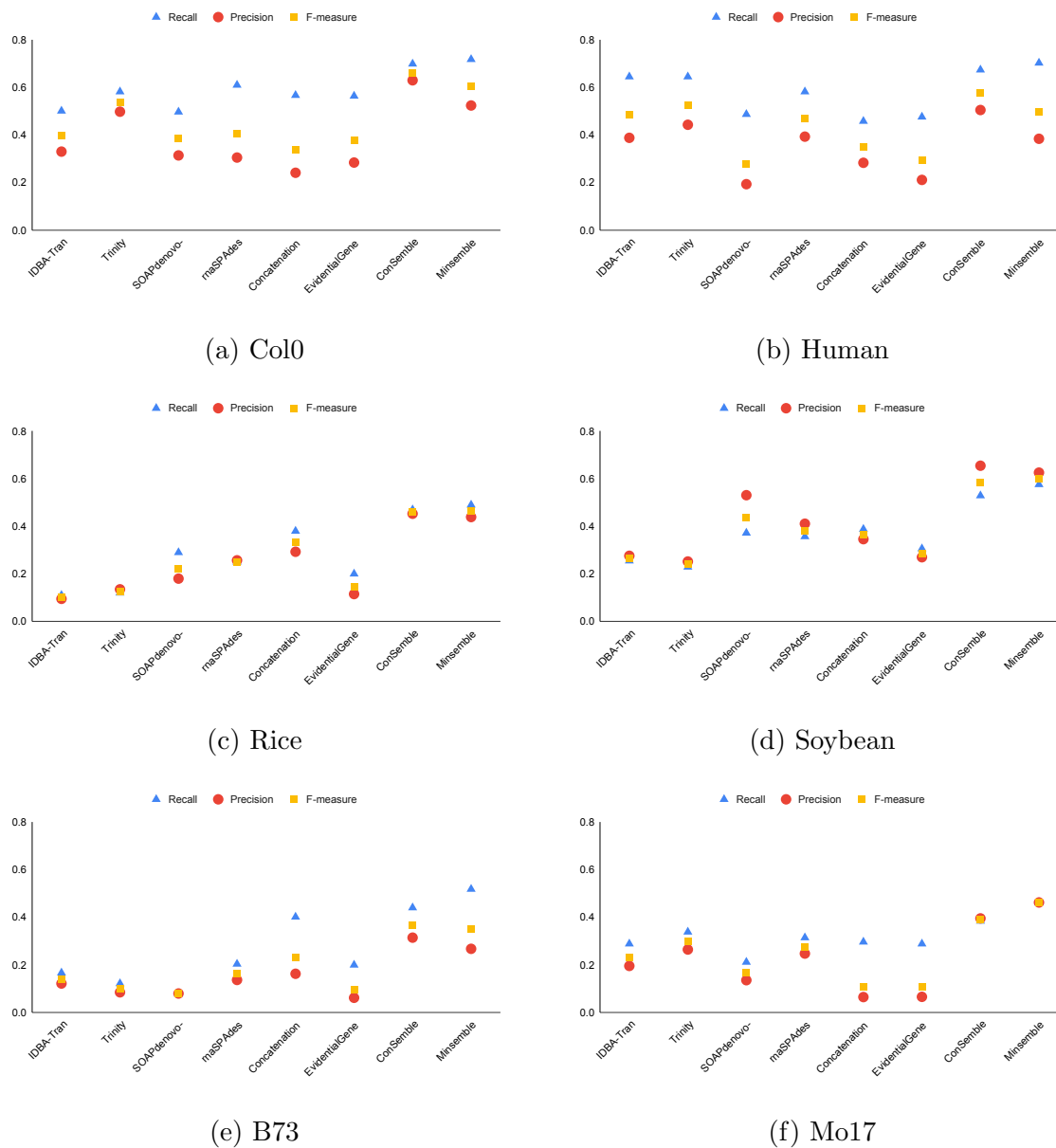


Figure 6.13: **Transcriptome assembler performance at the gene level.** Performance was compared among four *de novo* and four ensemble assembly methods. A gene is said to be correctly assembled if at least one of its isoform sequences is correctly assembled. The detailed performance metrics are given in Supplementary Tables A.9 - A.14.

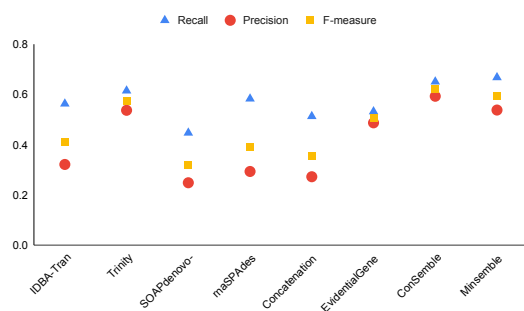
6.6.6.3 Performance of transcriptome assembly for the single-isoform genes

More than 90% of the genes for the Rice, Soybean, and *Z. mays* Mo17 datasets, more than 70% of the genes for the *A. thaliana* Col0 and *Z. mays* B73 datasets, and more than 50% of the genes for the Human dataset are single isoform genes (see Supplementary Table A.1). Therefore, we examined the assembler performance when only single-isoform genes are considered. The Recall, Precision, and F-measure scores of all assemblers are shown in Figure 6.14 and Supplementary Tables A.15 - A.20.

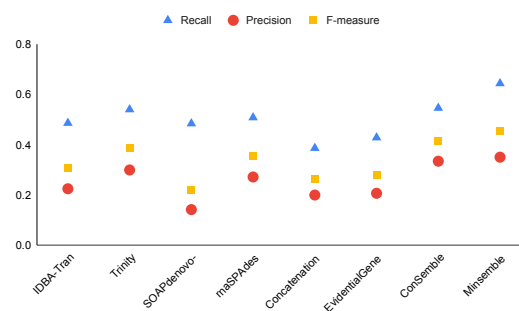
Minsemble had higher Recall values compared to all other assemblers, indicating that it identified single-isoform genes more correctly than other assemblers. Only for the *A. thaliana* Col0 and Soybean datasets, Precision scores of ConSemble were higher than those of Minsemble. However, Minsemble showed F-measure scores higher than all other assemblers for all the datasets except for the *A. thaliana* Col0 dataset.

6.6.6.4 Performance of transcriptome assembly for the multiple-isoform genes

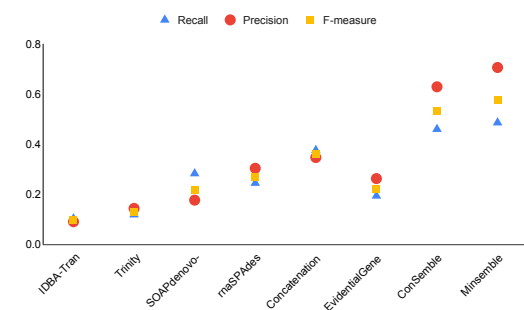
We next compared the performance of all transcriptome assemblers to evaluate their ability to reconstruct the isoforms. The *A. thaliana* Col0, *Z. mays* B73, and Human datasets have more than 20% of the genes that contain multiple isoforms (see Supplementary Table A.1). The *Z. mays* B73 dataset is the most complex with ~ 40 genes having ≥ 10 isoforms and two genes having more than 20 isoforms. For multiple-isoform genes, a gene is considered to be correctly assembled if all of its isoforms are correctly assembled. The Recall, Precision, and F-measure scores were compared among the four *de novo* and four ensemble assemblers (Figure 6.15 and Supplementary Tables A.21-A.26). Except for the Human dataset, Minsemble outperformed all



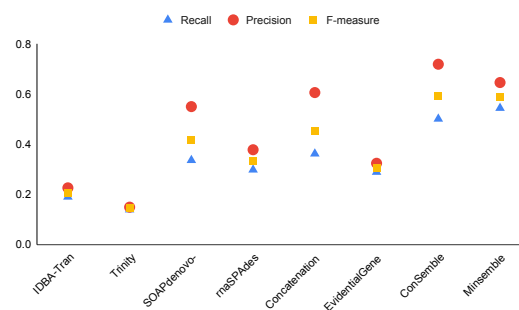
(a) Col0



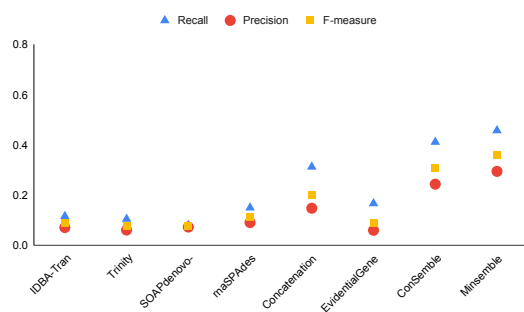
(b) Human



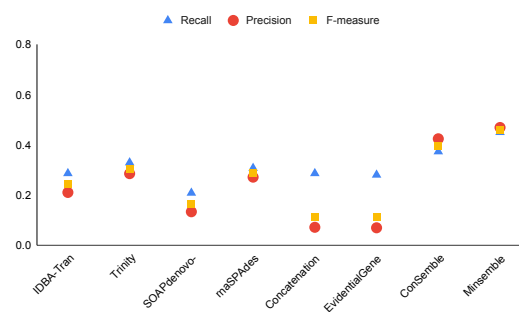
(c) Rice



(d) Soybean



(e) B73



(f) Mo17

Figure 6.14: **Transcriptome assembler performance for the single-isoform genes.** Performance was compared among four *de novo* and four ensemble assembly methods. The detailed performance metrics are given in Supplementary Tables A.15 - A.20.

other assemblers by reconstructing all isoforms.

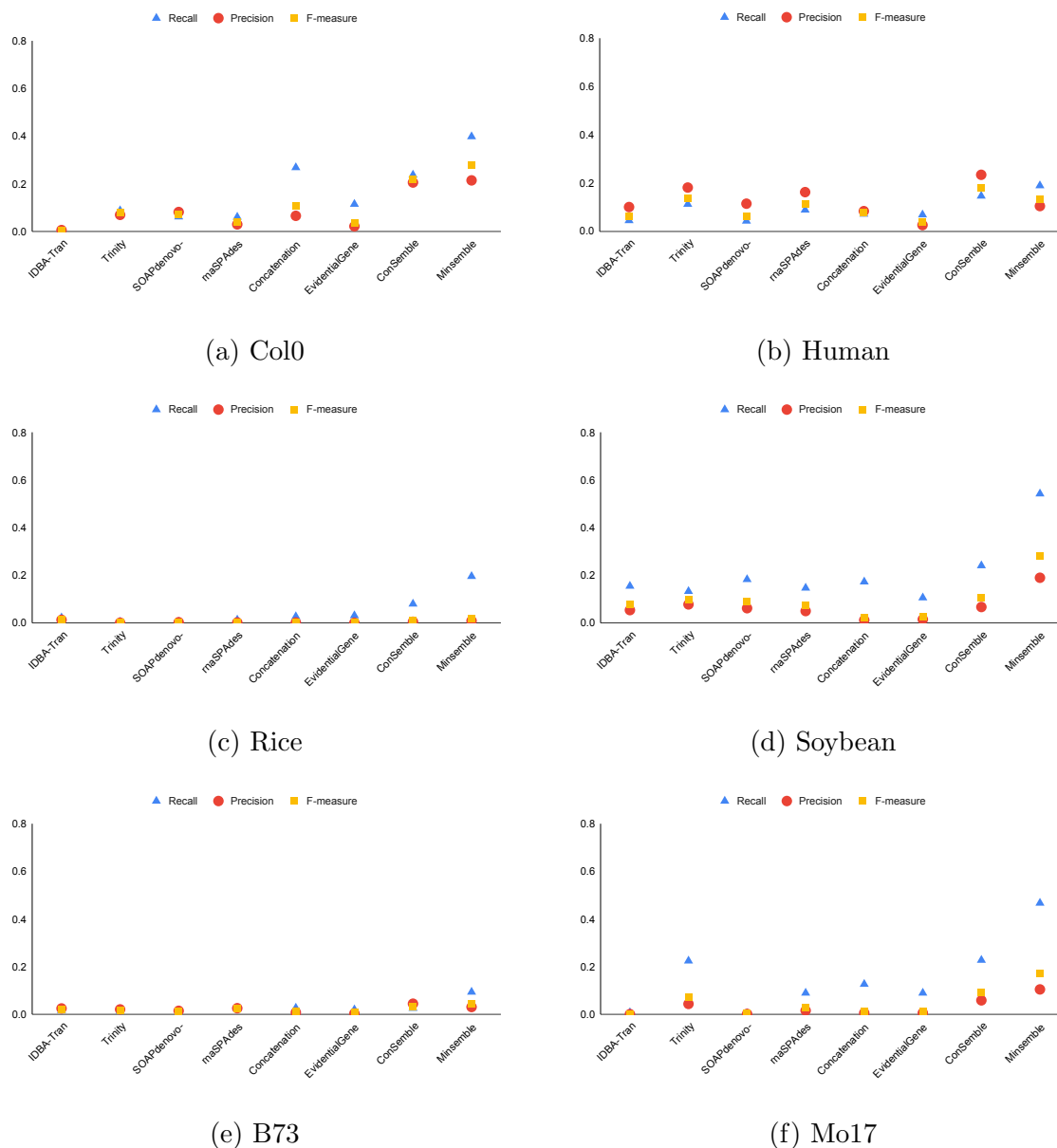


Figure 6.15: **Transcriptome assembler performance for the multiple-isoform genes.** Performance was compared among four *de novo* and four ensemble assembly methods. A gene is considered to be correctly assembled if all of its isoform sequences are correctly assembled. The detailed performance metrics are given in Supplementary Tables A.21 - A.26.

6.7 Conclusion

In this chapter, we performed comparative analysis among various transcriptome assembly methods using simulated benchmark datasets. We showed that how the availability of the high-quality reference genomes affects the transcriptome assembly performance by the genome-guided approach. When such reference genomes are not available, as in the case for non-model organisms, *de novo* assemblers can achieve good performance. However, challenges due to the isoform complexity, polyploidy, as well as optimal parameter selection remain. The most significant parameter in de Bruijn graph-based *de novo* assembly methods is the k -mer size. Ensemble approaches take advantage of pooling the *de novo* assemblies based on different methods as well as multiple k -mer to increase the number of correct contigs without accumulating incorrect contigs. Among the four ensemble methods compared in this study, our two methods (ConSemble and Minsemble), both based on the consensus approach, performed the best for all benchmark datasets tested. We also note the importance of the simulated benchmark datasets for assessment and improvement of the performance of transcriptome assembly. Our new approach, Minsemble, uses a novel clustering technique to cluster the potential isoforms and then selects contigs from clusters for the final assembly. A contig is selected from a cluster if it is 99% similar with at least two other contigs generated by two different assemblers. The current version of Minsemble recovers the highest number of true positives when compared with other *de novo* and ensemble approaches. At the same time, the clustering of contigs produced by Minsemble is the advantage that is not available in any other assemblers. The clustering algorithm (MinIsoClust) that is used in Minsemble can also be independently used for clustering the potential isoforms in any assembly. As we demonstrated, it can be used to perform better analysis at the gene level. The

future work remains how to reduce the false positives further and bring especially the Precision and F-measure scores to the ConSemble level.

Chapter 7

Conclusion and future works

In this dissertation, we developed several novel and efficient algorithms for some bioinformatics problems. The k -mer counting and its frequency estimation is useful in many bioinformatics applications such as metagenome analysis, and genome and transcriptome assembly. We used a streaming algorithm to estimate the frequency counts of k -mers efficiently. The streaming model used in our problem not only guarantees the upper and lower bounds of approximation, but also time and space efficiency. Conserved non-coding sequences (or elements) play an important role in regulating gene expression. Therefore, identification of these elements are important in functional genomics. These regions are conserved across different genomes and can be considered as variable-length k -mers. We proposed two different algorithms for identifying CNSs in plants and animals using suffix tree and minwise hashing, respectively. The suffix-tree based algorithm performs well when identification of only exact matched CNSs are required. However, minhash and LSH based approached performed better for identifying longer CNSs that are both exactly matched and with mismatches. The minhash approach was also used for two other problems in our dissertation study: isoform clustering and improved ensemble transcriptome assembly. The implementaion of these algorithms using parallel programming and threading remains as our future works.

Bibliography

- [1] Open reading frame finder. <https://www.ncbi.nlm.nih.gov/orffinder/>.
- [2] I. Akogwu, N. Wang, C. Zhang, and P. Gong. A comparative study of k -spectrum-based error correction methods for next-generation sequencing data analysis. *Human Genomics*, 10(2):20, 2016.
- [3] S. Altschul, T. L. Madden, A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.
- [4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.
- [5] A. Assefa, J. Vandesompele, and O. Thas. SPsimseq: semi-parametric simulation of bulk and single-cell rna-sequencing data. *Bioinformatics*, 36:3276–3278, 2020.
- [6] P. Audano and F. Vannberg. KAnalyze: a fast versatile pipelined k -mer toolkit. *Bioinformatics*, 30(14):2070–2072, 2014.
- [7] L. A. K. Ayad, S. P. Pissis, and D. Polychronopoulos. CNEFinder: finding conserved non-coding elements in genomes. *Bioinformatics*, 34(17):i743–i747, 2018.

- [8] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002, Proceedings*, pages 1–10, 2002.
- [9] P. Bardou, J. Mariette, F. Escudié, C. Djemiel, and C. Klopp. jvenn: an interactive Venn diagram viewer. *BMC Bioinformatics*, 15(1):293, 2014.
- [10] L. Baxter et al. Conserved noncoding sequences highlight shared components of regulatory networks in dicotyledonous plants. *The Plant Cell*, 24(10):3949–3965, 2012.
- [11] S. Behera, X. Lai, J. C. Schnable, and J. S. Deogun. DiCE: Discovery of conserved noncoding sequences efficiently. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 79–82, 2017.
- [12] S. Behera, A. Voshall, J. S. Deogun, and E. N. Moriyama. Performance comparison and an ensemble approach of transcriptome assembly. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2226–2228, 2017.
- [13] S. Behera, S. Gayen, J. S. Deogun, and N. V. Vinodchandran. KmerEstimate: A Streaming Algorithm for Estimating k -mer Counts with Optimal Space Usage. In *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB '18*, pages 438–447. ACM, 2018.
- [14] S. Behera, J. S. Deogun, and E. N. Moriyama. MinCNE: identifying conserved non-coding elements using min-wise hashing. In *Advances in Computer Vision and Computational-Biology*. Springer, 2020.

- [15] S. Behera, J. S. Deogun, and E. N. Moriyama. MinIsoClust: Isoform Clustering Using Minhash and Locality Sensitive Hashing. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '20, pages 1–7. ACM, 2020.
- [16] S. Behera, A. Voshall, K. Kapil, J. S. Deogun, and E. N. Moriyama. Minsemble: Isoform-clustering based ensemble approach. (*Manuscript under preparation*), 2020.
- [17] S. Behera, A. Voshall, and E. N. Moriyama. Plant transcriptome assembly: review and benchmarking. In *Bioinformatics*. Exon, Brisbane, 2020 (In Press).
- [18] S. Benidt and D. Nettleton. SimSeq: a nonparametric approach to simulation of RNA-sequence datasets. *Bioinformatics*, 31(13):2131–40, 2015.
- [19] P. Bieganski, J. Riedl, J. V. Cartis, and E. F. Retzel. Generalized suffix trees for biological sequence data: applications and implementation. In *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, volume 5, pages 35–44, 1994.
- [20] M. Blanchette et al. Aligning Multiple Genomic Sequences With the Threaded Blockset Aligner. *Genome research*, 14:708–722, 2004.
- [21] B. H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, 1970.
- [22] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Computer System Sciences*, 60(3):630–659, 2000.

- [23] E. Bushmanova, D. Antipov, A. Lapidus, V. Suvorov, and A. Prjibelski. rnaQUAST: a quality assessment tool for *de novo* transcriptome assemblies. *Bioinformatics*, 32(14):2210–2212, 2016.
- [24] E. Bushmanova, D. Antipov, A. Lapidus, and A. Prjibelski. rnaSPAdes: a *de novo* transcriptome assembler and its application to RNA-Seq data. *Giga-Science*, 8(9), 2019.
- [25] A. B. Carvalho, E. G. Dupim, and G. Goldstein. Improved assembly of noisy long reads by *k*-mer validation. *Genome research*, 26(12):1710–1720, 2016.
- [26] N. Cerveau and D. Jackson. Combining independent *de novo* assemblies optimizes the coding transcriptome for nonconventional model eukaryotic organisms. *BMC Bioinformatics*, 17(1):525, 2016.
- [27] S. Cha and D. Bird. Optimizing *k*-mer size using a variant grid search to enhance *de novo* genome assembly. *Bioinformation*, 12(2):36–40, 2016.
- [28] R. Chikhi and P. Medvedev. Informed and automated *k*-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014.
- [29] F. Y. L. Chin, H. C. M. Leung, M.-J. Lv, S.-M. Yiu, X.-G. Zhu, and Y. Peng. IDBA-tran: a more robust *de novo* de Bruijn graph assembler for transcriptomes with uneven expression levels. *Bioinformatics*, 29(13):i326–i334, 2013.
- [30] B. Chor, D. Horn, N. Goldman, Y. Levy, and T. Massingham. Genomic DNA *k*-mer spectra: models and modalities. *Genome Biology*, 10:R108–R108, 2009.
- [31] T. Christiani and R. Pagh. Set similarity search beyond minhash. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 1094–1107. ACM, 2017.

- [32] P. Compeau, P. A Pevzner, and G. Tesler. How to apply de Bruijn graphs to genome assembly. *Nature biotechnology*, 29:987–991, 2011.
- [33] G. Cormode, S. Muthukrishnan, and I. Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *Proceedings of the 31st international conference on Very large data bases*, pages 25–36. VLDB Endowment, 2005.
- [34] M. R. Crusoe et al. The khmer software package: enabling efficient nucleotide sequence analysis. *F1000Research*, 4:900, 2015.
- [35] N. M. Davidson and A. Oshlack. Corset: enabling differential gene expression analysis for *de novo* assembled transcriptomes. *Genome Biology*, 15(7):410, 2014.
- [36] S. Deorowicz, M. Kokot, S. Grabowski, and A. Debudaj-Grabysz. KMC 2: fast and resource-frugal *k*-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015.
- [37] S. Dimitrieva and P. Bucher. UCNEbase – a database of ultraconserved non-coding elements and genomic regulatory blocks. *Nucleic Acids Research*, 41(D1):D101–D109, 2012.
- [38] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.
- [39] A. Dousse, T. Junier, and E. M. Zdobnov. CEGA – a catalog of conserved elements from genomic alignments. *Nucleic Acids Research*, 44(D1):D96–D100, 2015.

- [40] D. A. Durai and M. H. Schulz. Informed k -mer selection for *de novo* transcriptome assembly. *Bioinformatics*, 32(11):1670–1677, 2016.
- [41] P. G. Engström, D. Fredman, and B. Lenhard. Ancora: a web resource for exploring highly conserved noncoding elements and their association with developmental regulatory genes. *Genome Biology*, 9:R34, 2007.
- [42] M. Erbert, S. Rechner, and M. Müller-Hannemann. Gerbil: a fast and memory-efficient k -mer counter with gpu-support. *Algorithms for Molecular Biology*, 12(1):9, 2017.
- [43] C. D. Fabbro, S. Scalabrin, M. Morgante, and F. M. Giorgi. An Extensive Evaluation of Read Trimming Effects on Illumina NGS Data Analysis. *PLoS ONE*, 8(12), 2013.
- [44] L. Florea and S. Salzberg. Genome-Guided Transcriptome Assembly in the Age of Next-Generation Sequencing. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 10(5):1234–1240, 2013.
- [45] A. C. Frazee, A. Jaffe, B. Langmead, and J. Leek. Polyester: simulating RNA-seq datasets with differential transcript expression. *Bioinformatics*, 31(17):2778–2784, 2015.
- [46] M. Freeling and S. Subramaniam. Conserved noncoding sequences (CNSs) in higher plants. *Current Opinion in Plant Biology*, 12(2):126–132, 2009.
- [47] D. Gerard. Data-based RNA-seq simulations by binomial thinning. *BMC Bioinformatics*, 21:206, 2020.
- [48] D. Gilbert. Genes of the pig, *Sus scrofa*, reconstructed with EvidentialGene. *PeerJ*, 7:e6374, 2019.

- [49] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.
- [50] S. Gnerre et al. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences of the United States of America*, 108(4):1513–1518, 2011.
- [51] E. Góngora-Castillo and C. Buell. Bioinformatics challenges in *de novo* transcriptome assembly using short read sequences in the absence of a reference genome sequence. *Natural product reports*, 30(4):490–500, 2013.
- [52] M. Grabherr et al. Full-length transcriptome assembly from RNA-Seq data without a reference genome. *Nature biotechnology*, 29(7):644–52, 2011.
- [53] T. Griebel, B. Zacher, P. Ribeca, E. Raineri, V. Lacroix, R. Guigó, and M. Sammeth. Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic Acids Research*, 40:10073–10083, 2012.
- [54] D. Gusfield. *Algorithms On Strings, Trees, and Sequences : Computer Science and Computational Biology.* ; Cambridge University Press, 1997.
- [55] J. J. Gutierrez-Gonzalez and D. F. Garvin. *de novo* Transcriptome Assembly in Polyploid Species. *Methods in molecular biology*, 1536:209–221, 2017.
- [56] S. Hafezqorani, C. Yang, T. Lo, K. M. Nip, R. L. Warren, and I. Birol. TransNanoSim characterizes and simulates nanopore RNA-sequencing data. *Giga-Science*, 9(6), 06 2020.
- [57] R. S. Harris. Improved pairwise alignment of genomic DNA. *Ph.D. Thesis, The Pennsylvania State University*, 2007.

- [58] B. Hass et al. *de novo* transcript sequence reconstruction from RNA-seq using the Trinity platform for reference generation and analysis. *Nature Protocols*, 8: 1494–1512, 2013.
- [59] A. Haudry, A. E. Platts, E. Vello, D. R. Hoen, M. Leclercq, R. J. Williamson, E. Forczek, Z. Joly-Lopez, J. G. Steffen, K. M. Hazzouri, et al. An atlas of over 90,000 conserved noncoding sequences provides insight into crucifer regulatory regions. *Nature genetics*, 45(8):891–898, 2013.
- [60] M. Hölzer and M. Marz. *de novo* transcriptome assembly: A comprehensive cross-species comparison of short-read RNA-Seq assemblers. *GigaScience*, 8, 2019.
- [61] M. Hozza, T. Vinař, and B. Brejová. How big is that genome? estimating genome size and coverage from k -mer abundance spectra. In *Proceedings of the 22Nd International Symposium on String Processing and Information Retrieval - Volume 9309*, SPIRE 2015, 2015.
- [62] P. H. Hsieh, Y. Oyang, and C. Y. Chen. Effect of *de novo* transcriptome assembly on transcript quantification. *Scientific Reports*, 9, 2019.
- [63] X. Huang, X. Chen, and P. Armbruster. Comparative performance of transcriptome assembly methods for non-model organisms. *BMC Genomics*, 17, 2016.
- [64] A. P. Hubert L. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

- [65] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC '98*, 1998.
- [66] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, 2010*, pages 41–52, 2010.
- [67] D. Kim, G. Pertea, C. Trapnell, H. Pimentel, R. Kelley, and S. L. Salzberg. TopHat2: accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions. *Genome Biology*, 14(4):R36, 2013.
- [68] D. Kim, J. M. Paggi, C. Park, C. Bennett, and S. Salzberg. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. *Nature Biotechnology*, 37:907–915, 2019.
- [69] M. Kokot, M. Dlugosz, and S. Deorowicz. KMC 3: counting and manipulating k -mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017.
- [70] D. Koslicki and H. Zabeti. Improving minhash via the containment index with applications to metagenomic analysis. *Appl. Math. Comput.*, 354:206–215, 2019.
- [71] S. Kovaka, A. V. Zimin, G. Pertea, R. Razaghi, S. L. Salzberg, and M. Pertea. Transcriptome assembly from long-read rna-seq alignments with stringtie2. *Genome Biology*, 20, 2019.
- [72] E. Kriventseva, D. Kuznetsov, F. Tegenfeldt, M. Manni, R. Dias, F. R. Simão, and E. Zdobnov. Orthodb v10: sampling the diversity of animal, plant, fungal,

- protist, bacterial and viral genomes for evolutionary and functional annotations of orthologs. *Nucleic Acids Research*, 47:D807–D811, 2019.
- [73] S. Kurtz, A. Narechania, J. Stein, and D. Ware. A new method to compute k -mer frequencies and its application to annotate large repetitive plant genomes. *BMC Genomics*, 9:517–517, 2008.
- [74] X. Lai, S. Behera, Z. Liang, Y. Lu, J. S. Deogun, and J. C. Schnable. STAG-CNS: An Order-Aware Conserved Noncoding Sequences Discovery Tool for Arbitrary Numbers of Species. *Molecular Plant*, 10(7):990–999, 2017.
- [75] B. Lau, M. Mohiyuddin, J. C. Mu, L. Fang, N. Asadi, C. Dallett, and H. Y. K. Lam. LongISLND: in silico sequencing of lengthy and noisy datatypes. *Bioinformatics*, 32:3829–3832, 2016.
- [76] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. 2014.
- [77] B. Li and C. N. Dewey. RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, 12:323–323, 2011.
- [78] B. Li, N. Fillmore, Y. Bai, M. Collins, J. Thomson, R. Stewart, and C. N. Dewey. Evaluation of *de novo* transcriptome assemblies from RNA-Seq data. *Genome Biology*, 15, 2014.
- [79] R. Li et al. *de novo* assembly of human genomes with massively parallel short read sequencing. *Genome research*, 20(2):265–272, 2010.

- [80] W. Li and A. Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, 22(13):1658–1659, 2006.
- [81] Y. Li and X. Yan. MSPKmerCounter: A Fast and Memory Efficient Approach for k -mer Counting. *CoRR*, abs/1505.06550, 2014.
- [82] M. Lipovsky, T. Vinar, and B. Brejova. Approximate Abundance Histograms and Their Use for Genome Size Estimation. In *Proceedings of the 17th Conference on Information Technologies - Applications and Theory, ITAT 2017*, pages 27–34, 2017.
- [83] B. Liu, Y. Shi, J. Yuan, X. Hu, H. Zhang, N. Li, Z. Li, Y. Chen, D. Mu, and W. Fan. Estimation of genomic characteristics by analyzing k -mer frequency in *de novo* genome projects. *arXiv: Genomics*, 2013.
- [84] S. Liu et al. Unbiased k -mer Analysis Reveals Changes in Copy Number of Highly Repetitive Sequences During Maize Domestication and Improvement. *Scientific Reports*, 7:42444, 2017.
- [85] V. Lomonaco, R. Martoglia, F. Mandreoli, L. Anderlucci, W. Emmett, S. Biciato, and C. Taccioli. UCbase 2.0: ultraconserved sequences database (2014 update). *Database*, 2014.
- [86] N. López-Bigas, B. Audit, C. A. Ouzounis, G. L. Parra, and R. Guigó. Are splicing mutations the most frequent cause of hereditary disease? *FEBS letters*, 579(9):1900–1903, 2005.

- [87] M. Love, J. Hogenesch, and R. Irizarry. Modeling of RNA-seq fragment sequence bias reduces systematic errors in transcript abundance estimation. *Nature biotechnology*, 34:1287–1291, 2016.
- [88] R. Luo et al. SOAPdenovo2: an empirically improved memory-efficient short-read *de novo* assembler. *GigaScience*, 1:18, 2012.
- [89] K. Mahmood, J. Orabi, P. S. Kristensen, P. Sarup, L. N. Jørgensen, and A. Jahoor. *de novo* transcriptome assembly, functional annotation, and expression profiling of rye (*Secale cereale* L.) hybrids inoculated with ergot (*Claviceps purpurea*). *Scientific Reports*, 10, 2020.
- [90] A. A. Mamun, S. Pal, and S. Rajasekaran. KCMBT: a k -mer Counter based on Multiple Burst Trees. *Bioinformatics*, 32(18):2783–2790, 2016.
- [91] S. Marcus, H. Lee, and M. C. Schatz. SplitMEM: A Graphical Algorithm for Pan-Genome Analysis with Suffix Skips. *Bioinformatics*, 30(24):3476–3483, 2014.
- [92] L. Maretty, J. A. Sibbesen, and A. Krogh. Bayesian transcriptome assembly. *Genome Biology*, 15(10):501, 2014.
- [93] J. Martin and Z. Wang. Next-generation transcriptome assembly. *Nature Reviews Genetics*, 12:671–682, 2011.
- [94] G. Marçais and C. Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k -mers. *Bioinformatics*, 27(6):764–770, 2011.
- [95] P. Melsted and B. V. Halldorsson. KmerStream: streaming algorithms for k -mer abundance estimation. *Bioinformatics*, 30(24):3541–3547, 2014.

- [96] P. Melsted and J. K. Pritchard. Efficient counting of k -mers in dna sequences using a bloom filter. *BMC Bioinformatics*, 12(1):1–7, 2011.
- [97] H. Mohamadi, J. Chu, B. P. Vandervalk, and I. Birol. ntHash: recursive nucleotide hashing. *Bioinformatics*, 32(22):3492–3494, 2016.
- [98] H. Mohamadi, H. Khan, and I. Birol. ntCard: a streaming algorithm for cardinality estimation in genomics data. *Bioinformatics*, 33(9):1324–1330, 2017.
- [99] S. Muthukrishnan. Data Streams: Algorithms and Applications. *Found. Trends Theor. Comput. Sci.*, 1(2), Aug. 2005.
- [100] D. Y. Nishimura et al. Comparative genomics and gene expression analysis identifies bbs9, a new bardet-biedl syndrome gene. *American Journal of Human Genetics*, 77(6):1021–1033, 2005.
- [101] B. D. Ondov, T. J. Treangen, P. Melsted, A. B. Mallonee, N. H. Bergman, S. Koren, and A. M. Phillippy. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17, 2016.
- [102] S. T. O’Neil and S. Emrich. Assessing *de novo* transcriptome assembly metrics for consistency and utility. *BMC Genomics*, 14:465–465, 2012.
- [103] Y. Ono, K. Asai, and M. Hamada. PBSIM: PacBio reads simulator - toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2013.
- [104] A. Oshlack, M. Robinson, and M. Young. From RNA-seq reads to differential expression results. *Genome Biology*, 11:220–220, 2010.
- [105] F. Ozsolak and P. Milos. RNA sequencing: advances, challenges and opportunities. *Nature Reviews Genetics*, 12:87–98, 2011.

- [106] G. P. sparsepp. <https://github.com/greg7mdp/sparsepp>, 2016.
- [107] Q. Pan, O. Shai, L. J. Lee, B. J. Frey, and B. J. Blencowe. Deep surveying of alternative splicing complexity in the human transcriptome by high-throughput sequencing. *Nature Genetics*, 40:1413–1415, 2008.
- [108] P. Pandey, M. A. Bender, R. Johnson, and R. Patro. Squeakr: an exact and approximate k -mer counting system. *Bioinformatics*, 34(4):568–575, 2018.
- [109] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12:2825–2830, 2011.
- [110] J. Pell et al. Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. *Proceedings of the National Academy of Sciences of the United States of America*, 109(33):13272–13277, 2012.
- [111] J. Persampieri, D. I. Ritter, D. Lees, J. Lehoczky, Q. Li, S. Guo, and J. H. Chuang. cneViewer: a database of conserved non-coding elements for studies of tissue-specific gene regulation. *Bioinformatics*, 24(20):2418–2419, 2008.
- [112] M. Pertea, G. M. Pertea, C. M. Antonescu, T.-C. Chang, J. T. Mendell, and S. L. Salzberg. StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nat Biotech*, 33(3):290–295, 2015.
- [113] D. Polychronopoulos, J. W. King, A. J. Nash, G. Tan, and B. Lenhard. Conserved non-coding elements: developmental gene regulation meets genome organization. *Nucleic Acids Research*, 45(22):12611–12624, 2017.

- [114] S. Rangavittal et al. RecoverY: k -mer based read classification for Y-chromosome specific sequencing and assembly. *Bioinformatics*, 34(7):1125–1131, 2017.
- [115] G. Rizk, D. Lavenier, and R. Chikhi. Dsk: k -mer counting with very low memory usage. *Bioinformatics*, 29(5):652–653, 2013.
- [116] E. Rodgers-Melnick et al. Open chromatin reveals the functional maize genome. *Proceedings of the National Academy of Sciences of the United States of America*, 113(22):E3177–E3184, 2016.
- [117] A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *EMNLP-CoNLL*, 2007.
- [118] R. S. Roy, D. Bhattacharya, and A. Schliep. Turtle: Identifying frequent k -mers with cache-efficient algorithms. *Bioinformatics*, 30(14):1950–1957, 2014.
- [119] K. Sahlin and P. Medvedev. *de novo* clustering of long-read transcriptome data using a greedy, quality-value based algorithm. In *RECOMB*, 2019.
- [120] S. Salzberg et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms. *Genome research*, 22(3):557–567, 2012.
- [121] J. Schnable, Y. Zang, and D. W.C. Ngu. Pan-grass syntenic gene set (sorghum referenced). *Figshare*, 2016. URL <https://dx.doi.org/10.6084/m9.figshare.3113488.v1>.
- [122] S. Schwartz, W. J. Kent, A. Smit, Z. Zhang, R. Baertsch, R. Hardison, D. Hausler, and W. Miller. Human-mouse alignments with BLASTZ. *Genome research*, 13:103–110, 2003.

- [123] M. Shao and C. Kingsford. Accurate assembly of transcripts through phase-preserving graph decomposition. *Nature Biotechnology*, 35, 2017.
- [124] A. Shrivastava and P. Li. Asymmetric minwise hashing for indexing binary inner products and set containment. In *WWW '15*, 2015.
- [125] F. R. Simão, R. Waterhouse, P. Ioannidis, E. Kriventseva, and E. Zdobnov. BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, 31(19):3210–2, 2015.
- [126] N. Sivadasan, R. Srinivasan, and K. Goyal. Kmerlight: fast and accurate k -mer abundance estimation. *CoRR*, abs/1609.05626, 2016.
- [127] R. D. Smith-Unna, C. Boursnell, R. Patro, J. Hibberd, and S. Kelly. TransRate: reference-free quality assessment of *de novo* transcriptome assemblies. *Genome research*, 26(8):1134–1144, 2016.
- [128] R. Stark, M. Grzelak, and J. Hadfield. RNA sequencing: the teenage years. *Nature Reviews Genetics*, 20:1–26, 2019.
- [129] M. Steinegger and J. Söding. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nature Biotechnology*, 35:1026–1028, 2017.
- [130] S. Stephen, M. Pheasant, I. V. Makunin, and J. S. Mattick. Large-scale appearance of ultraconserved elements in tetrapod genomes and slowdown of the molecular clock. *Molecular biology and evolution*, 25(2):402–408, 2008.
- [131] H. Tang, E. Lyons, B. Pedersen, J. C. Schnable, A. Paterson, and M. Freeling. Screening Synteny Blocks in Pairwise Genome Comparisons through Integer Programming. *BMC bioinformatics*, 12:102, 2011.

- [132] The 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, 2010.
- [133] B. C. Thomas, L. Rapaka, E. Lyons, B. Pedersen, and M. Freeling. Arabidopsis intragenomic conserved noncoding sequence. *PNAS*, 104(9):3348–3353, 2007.
- [134] A. Tiskin. Semi-local string comparison : algorithmic techniques and applications. *Mathematics in Computer Science*, 1(4):571–603, 2008.
- [135] C. Trapnell, B. A. Williams, G. Pertea, A. Mortazavi, G. Kwan, M. V. van Baren, S. Salzberg, B. Wold, and L. Pachter. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature biotechnology*, 28(5):511–515, 2010.
- [136] C. Trapnell, A. Roberts, L. Goff, G. Pertea, D. Kim, D. Kelley, H. Pimentel, S. Salzberg, J. Rinn, and L. Pachter. Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks. *Nature Protocols*, 7(3):562–578, 2012.
- [137] M. Tsompana and M. J. Buck. Chromatin accessibility: a window into the genome. *Epigenetics & chromatin*, 7(1):33, 2014.
- [138] G. Turco, J. C. Schnable, B. Pedersen, and M. Freeling. Automated conserved non-coding sequence (CNS) discovery reveals differences in gene content and promoter evolution among grasses. *Frontiers in plant science*, 4:170, 2013.
- [139] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
- [140] J. Van de Velde et al. A collection of conserved noncoding sequences to study gene regulation in flowering plants. *Plant Physiology*, 171(4):2586–2598, 2016.

- [141] N. Vijay, J. W. Poelstra, A. Künstner, and J. Wolf. Challenges and strategies in transcriptome assembly and differential gene expression quantification. A comprehensive in silico assessment of RNA-seq experiments. *Molecular Ecology*, 22, 2013.
- [142] A. Visel, S. Minovitsky, I. Dubchak, and L. A. Pennacchio. VISTA Enhancer Browser—a database of tissue-specific human enhancers. *Nucleic Acids Research*, 35(suppl_1):D88–D92, 2006.
- [143] A. Voshall and E. N. Moriyama. Next-generation transcriptome assembly: Strategies and performance analysis. In I. Y. Abdurakhmonov, editor, *Bioinformatics in the Era of Post Genomics and Big Data*, chapter 2. IntechOpen, Rijeka, 2018.
- [144] A. Voshall and E. N. Moriyama. Next-generation transcriptome assembly and analysis: impact of ploidy. *Methods*, 176:14–24, 2019.
- [145] A. Voshall, S. Behera, X. Li, X. Yu, K. Kapil, J. S. Deogun, J. Shanklin, E. Cahoon, and E. N. Moriyama. A consensus-based ensemble approach to improve *de novo* transcriptome assembly. *bioRxiv*, 2020. doi: 10.1101/2020.06.08.139964.
- [146] D. L. Wheeler, D. Church, S. Federhen, A. Lash, T. L. Madden, J. Pontius, G. Schuler, L. Schriml, E. Sequeira, T. Tatusova, and L. Wagner. Database resources of the National Center for Biotechnology. *Nucleic acids research*, 31(1):28–33, 2003.
- [147] R. Wick. Badread: simulation of error-prone long reads. *J. Open Source Softw.*, 4:1316, 2019.

- [148] A. Woolfe, D. K. Goode, J. E. Cooke, H. Callaway, S. F. Smith, P. J. Snell, G. McEwen, and G. Elgar. CONDOR: a database resource of developmentally associated conserved non-coding elements. *BMC Developmental Biology*, 7:100, 2007.
- [149] Y. Xie, G. Wu, J. Tang, R. Luo, J. Patterson, S. Liu, W. Huang, G. He, S. Gu, S. Li, X. Zhou, T. W. Lam, Y. Li, X. Xu, and G. K.-S. Wong. SOAPdenovo-Trans: *de novo* transcriptome assembly with short RNA-Seq reads. *Bioinformatics*, 30(12):1660–1666, 2014.
- [150] X. Yang, S. P. Chockalingam, and S. Aluru. A survey of error-correction methods for next-generation sequencing. *Briefings in bioinformatics*, 14(1):56–66, 2013.
- [151] T. Yu, Z. Mu, Z. Fang, X. Liu, X. Gao, and J. Liu. TransBorrow: genome-guided transcriptome assembly by borrowing assemblies from different assemblers. *Genome research*, 2020.
- [152] B. Zacher, E. Raineri, P. Ribeca, R. Guigó, T. Griebel, V. Lacroix, and M. Sammeth. Modelling and simulating generic RNA-Seq experiments with the flux simulator. *Nucleic Acids Research*, 40(20):10073–10083, 2012.
- [153] D. R. Zerbino and E. Birney. Velvet: algorithms for *de novo* short read assembly using de bruijn graphs. *Genome research*, 18 5:821–9, 2008.
- [154] Q. Zhang, J. Pell, R. Canino, C. Howe, and T. Brown. These Are Not the k -mers You Are Looking For: Efficient Online k -mer Counting Using a Probabilistic Data Structure. *PLoS ONE*, 9(7), 2014.

- [155] W. Zhang et al. High-resolution mapping of open chromatin in the rice genome. *Genome Research*, 22(1):151–162, 2012.
- [156] M. Zhao, D. Liu, and H. Qu. Systematic review of next-generation sequencing simulators: computational tools, features and perspectives. *Briefings in Functional Genomics*, 16:121–128, 2017.
- [157] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. LSH Ensemble: Internet-Scale Domain Search. *Proc VLDB Endowment*, 9(12):1185–1196, 2016.
- [158] J. Zook et al. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific Data*, 3:160025, 2016.
- [159] M. Šošić and M. Šikić. Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 33(9):1394–1395, 2017.

Appendix A

Table A.1: Isoform distribution

# isoforms per gene	No0	Col0	Rice	Soybean	B73	Mo17	Human
1	18,875	9,502	10,836	15,925	8,455	16,335	8,575
2		1854	213	934	1,710	437	2,365
3		465	11	103	656	71	796
4		142		16	278	9	288
5		40		6	126	3	69
6		14		2	64	1	31
7		4		1	39		16
8		1			32		5
9					16		3
10					10		1
11					7		2
12					6		2
13					5		1
14					2		
15					1		
16					1		
17					2		
19					1		
20					2		
Total # genes	18,875	12,023	11,060	16,987	11,413	16,856	16,856
% single-isoform gene	100%	79.03%	97.97%	93.75%	74.08%	96.91%	50.87%
Total # transcripts	18,875	15,502	11,294	18,215	17,108	17,479	17,669

Table A.2: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *A. thaliana* No0 dataset at the transcript level^a

Assembler	Total ^b	TP	FP	FN	C/I	Accuracy*	Recall	Precision	F-measure
IDBA-Tran	22,768	8,353	14,415	10,522	0.579	0.251	0.443	0.367	0.401
Trinity	29,773	11,132	18,641	7,743	0.597	0.297	0.590	0.374	0.458
SOAPdenovo-trans	23,476	12,073	11,403	6,802	1.059	0.399	0.640	0.514	0.570
rnaSPAdes	27,664	10,045	17,619	8,830	0.57	0.275	0.532	0.363	0.432
Concatenation	73,539	4,931	68,608	13,944	0.072	0.056	0.261	0.067	0.107
EvidentialGene	71,161	9,444	61,717	9,431	0.153	0.117	0.500	0.133	0.210
ConSemble	20,298	13,371	6,927	5,504	1.93	0.518	0.708	0.659	0.683
Minsemble	40,528	13,932	26,596	4,943	0.524	0.306	0.738	0.344	0.469

^a Total number of the benchmark transcripts is 18,875.

^b Total number of assembled contigs.

Table A.3: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *A. thaliana* Col0 dataset at the transcript level^a

Assembler	Total ^b	TP	FP	FN	C/I	Accuracy*	Recall	Precision	F-measure
IDBA-Tran	20,447	6,054	14,393	9,454	0.421	0.202	0.390	0.296	0.337
Trinity	21,371	7,324	14,047	8,184	0.521	0.248	0.472	0.343	0.397
SOAPdenovo-trans	19,409	9,309	10,100	6,199	0.922	0.364	0.600	0.480	0.533
rnaSPAdes	31,494	7,599	23,895	7,909	0.318	0.193	0.490	0.241	0.323
Concatenation	46,033	7,527	38,506	7,981	0.195	0.139	0.485	0.164	0.245
EvidentialGene	64,007	7,908	56,099	7,600	0.141	0.110	0.510	0.124	0.199
ConSemble	17,309	9,245	8,064	6,263	1.146	0.392	0.596	0.534	0.563
Minsemble	32,150	10,199	21,951	5,309	0.465	0.272	0.658	0.317	0.428

^a Total number of benchmark transcripts is 15,508.

^b Total number of assembled contigs.

Table A.4: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Rice dataset at the transcript level^a

Assembler	Total ^b	TP	FP	FN	C/I	Accuracy*	Recall	Precision	F-measure
IDBA-Tran	13,151	3,450	9,701	7,844	0.356	0.164	0.306	0.262	0.282
Trinity	18,000	4,157	13,843	7,137	0.3	0.165	0.368	0.231	0.284
SOAPdenovo-trans	10,508	5,066	5,442	6,228	0.931	0.303	0.449	0.482	0.465
rnaSPAdes	13,182	5,386	7,796	5,908	0.691	0.282	0.477	0.409	0.440
Concatenation	57,541	2,204	55,337	9,090	0.04	0.033	0.195	0.038	0.064
EvidentialGene	33,920	4,279	29,641	7,015	0.144	0.105	0.379	0.126	0.189
ConSemble	14,922	5,605	9,317	5,689	0.602	0.272	0.496	0.376	0.428
Minsemble	25,224	5,840	19,384	5,454	0.301	0.19	0.517	0.232	0.320

^a Total number of benchmark transcripts is 11,294.

^b Total number of assembled contigs.

Table A.5: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Soybean dataset at the transcript level^a

Assembler	Total ^b	TP	FP	FN	C/I	Accuracy*	Recall	Precision	F-measure
IDBA-Tran	33,243	6,043	27,200	12,172	0.222	0.133	0.332	0.182	0.235
Trinity	52,700	5,462	47,238	12,753	0.116	0.083	0.3	0.104	0.154
SOAPdenovo-trans	24,346	8,424	15,922	9,791	0.529	0.247	0.462	0.346	0.396
rnaSPAdes	23,686	8,120	15,566	10,095	0.522	0.24	0.446	0.343	0.388
Concatenation	48,442	6,819	41,623	11,396	0.164	0.114	0.374	0.141	0.205
EvidentialGene	61,136	6,832	54,304	11,383	0.126	0.094	0.375	0.112	0.172
ConSemble	18,525	9,512	9,013	8,703	1.055	0.349	0.522	0.513	0.518
Minsemble	34,642	10,200	24,442	8,015	0.417	0.239	0.56	0.294	0.386

^a Total number of benchmark transcripts is 18,215.^b Total number of assembled contigs.Table A.6: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *Z. mays* B73 dataset at the transcript level^a

Assembler	Total ^b	TP	FP	FN	C/I	Accuracy*	Recall	Precision	F-measure
IDBA-Tran	24,603	4,067	20,536	13,041	0.198	0.108	0.238	0.165	0.195
Trinity	27,403	3,127	24,276	13,981	0.129	0.076	0.183	0.114	0.141
SOAPdenovo-trans	22,327	5,736	16,591	11,372	0.346	0.170	0.335	0.257	0.291
rnaSPAdes	23,764	4,012	19,752	13,096	0.203	0.109	0.235	0.169	0.196
Concatenation	113,689	2,418	111,271	14,690	0.022	0.019	0.141	0.021	0.037
EvidentialGene	74,811	5,389	69,422	11,719	0.078	0.062	0.315	0.072	0.117
ConSemble	18,150	5,212	12,938	11,896	0.403	0.173	0.305	0.287	0.296
Minsemble	35,242	6,598	28,644	10,510	0.230	0.144	0.386	0.187	0.252

^a Total number of benchmark transcripts is 17,108.^b Total number of assembled contigs.Table A.7: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *Z. mays* Mo17 dataset at the transcript level^a

Assembler	Total ^b	TP	FP	FN	C/I	Accuracy*	Recall	Precision	F-measure
IDBA-Tran	24,916	4,881	20,035	12,598	0.244	0.13	0.279	0.196	0.23
Trinity	26,257	3,582	22,675	13,897	0.158	0.089	0.205	0.136	0.164
SOAPdenovo-trans	21,537	5,857	15,680	11,622	0.374	0.177	0.335	0.272	0.3
rnaSPAdes	21,469	5,377	16,092	12,102	0.334	0.16	0.308	0.251	0.276
Concatenation	71,471	2,863	68,608	14,616	0.042	0.033	0.164	0.040	0.064
EvidentialGene	76,496	5,090	71,406	12,389	0.071	0.057	0.291	0.067	0.108
ConSemble	16,406	6,642	9,764	10,837	0.68	0.244	0.380	0.405	0.392
Minsemble	29,561	7,092	22,469	10,387	0.316	0.178	0.406	0.240	0.302

^a Total number of benchmark transcripts is 17,479.^b Total number of assembled contigs.

Table A.8: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Human dataset at the transcript level^a

Assembler	Total ^b	TP	FP	FN	C/I	Accuracy*	Recall	Precision	F-measure
IDBA-Tran	20,954	6,154	14,800	11,515	0.42	0.19	0.348	0.294	0.319
SOAPdenovo	22,005	5,933	16,072	11,736	0.37	0.176	0.336	0.27	0.299
Trinity	21,278	8,764	12,514	8,905	0.7	0.29	0.496	0.412	0.450
rnaSPAdes	21,244	7,637	13,607	10,032	0.56	0.244	0.432	0.359	0.393
EvidentialGene	65,587	8,680	56,907	8,989	0.15	0.116	0.491	0.132	0.209
Concatenation	45,180	7,793	37,387	9,876	0.21	0.142	0.441	0.172	0.248
ConSemble	19,509	9,200	10,309	8,469	0.89	0.329	0.521	0.472	0.495
Minsemble	32,071	9,871	22,200	7,798	0.44	0.248	0.559	0.308	0.397

^a Total number of the benchmark transcripts is 17,669.

^b Total number of assembled contigs.

Table A.9: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *A. thaliana* Col0 dataset at the gene level^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	18,234	6,040	5,983	12,194	0.502	0.331	0.249	0.399
Trinity	14,036	7,007	5,016	7,029	0.583	0.499	0.368	0.538
SOAPdenovo-trans	19,016	5,987	6,036	13,029	0.498	0.315	0.239	0.386
rnaSPAdes	24,005	7,350	4,673	16,655	0.611	0.306	0.256	0.408
Concatenation	28,195	6,827	5,196	21,368	0.568	0.242	0.204	0.339
EvidentialGene	23,826	6,794	5,229	17,032	0.565	0.285	0.234	0.379
ConSemble	13,346	8,415	3,608	4,931	0.700	0.631	0.496	0.663
Minsemble	16,458	8,643	3,380	7,815	0.719	0.525	0.436	0.607

^a Total number of benchmark genes is 12,203.

^b Total number of clusters generated by MinIsoClust.

Table A.10: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Human dataset at the gene level^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	20,170	7,853	4,305	12,317	0.646	0.389	0.321	0.486
Trinity	17,685	7,853	4,305	9,832	0.646	0.444	0.357	0.526
SOAPdenovo-trans	30,539	5,936	6,222	24,603	0.488	0.194	0.161	0.278
rnaSPAdes	17,993	7,089	5,069	10,904	0.583	0.394	0.307	0.470
Concatenation	19,676	5,580	6,578	14,096	0.459	0.284	0.213	0.351
EvidentialGene	27,426	5,804	6,354	21,622	0.477	0.212	0.172	0.293
ConSemble	16,224	8,206	3,952	8,018	0.675	0.506	0.407	0.578
Minsemble	22,242	8,565	3,593	13,677	0.704	0.385	0.332	0.498

^a Total number of benchmark genes is 12,158.

^b Total number of clusters generated by MinIsoClust.

Table A.11: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Rice dataset at the gene level^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	12,665	1,209	11,456	9,851	0.109	0.095	0.054	0.102
Trinity	9,696	1,311	8,385	9,749	0.119	0.135	0.067	0.126
SOAPdenovo-trans	17,637	3,171	14,466	7,889	0.287	0.180	0.124	0.221
rnaSPAdes	10,580	2,720	7,860	8,340	0.246	0.257	0.144	0.251
Concatenation	14,527	4,250	10,277	6,810	0.384	0.293	0.199	0.332
EvidentialGene	19,169	2,197	16,972	8,863	0.199	0.115	0.078	0.145
ConSemble	11,405	5,165	6,240	5,895	0.467	0.453	0.299	0.460
Minsemble	12,364	5,433	6,931	5,627	0.491	0.439	0.302	0.464

^a Total number of benchmark genes is 11,060.

^b Total number of clusters generated by MinIsoClust.

Table A.12: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Soybean dataset at the gene level^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	15,642	4,325	11,317	12,662	0.255	0.276	0.153	0.265
Trinity	15,432	3,890	11,542	13,097	0.229	0.252	0.136	0.240
SOAPdenovo-trans	11,908	6,321	5,587	10,666	0.372	0.531	0.280	0.438
rnaSPAdes	14,789	6,071	8,718	10,916	0.357	0.411	0.236	0.382
Concatenation	19,059	6,591	12,468	10,396	0.388	0.346	0.224	0.366
EvidentialGene	19,247	5,191	14,056	11,796	0.306	0.270	0.167	0.287
ConSemble	13,737	8,991	4,746	7,996	0.529	0.655	0.414	0.585
Minsemble	15,623	9,777	5,846	7,210	0.576	0.626	0.428	0.600

^a Total number of benchmark genes is 16,987.

^b Total number of clusters generated by MinIsoClust.

Table A.13: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *Z. mays* B73 dataset at the gene level^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	15,642	1,903	13,739	9,510	0.167	0.122	0.076	0.141
Trinity	16,287	1,389	14,898	10,024	0.122	0.085	0.053	0.100
SOAPdenovo-trans	11,098	891	10,207	10,522	0.078	0.080	0.041	0.079
rnaSPAdes	16,995	2,327	14,668	9,086	0.204	0.137	0.089	0.164
Concatenation	28,136	4,584	23,552	6,829	0.402	0.163	0.131	0.232
EvidentialGene	36,765	2,283	34,482	9,130	0.200	0.062	0.050	0.095
ConSemble	16,003	5,033	10,970	6,380	0.441	0.315	0.225	0.367
Minsemble	22,127	5,926	16,201	5,487	0.519	0.268	0.215	0.353

^a Total number of benchmark genes is 11,413.

^b Total number of clusters generated by MinIsoClust.

Table A.14: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for *Z. mays* Mo17 dataset at the gene level^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	24,916	4,875	20,041	11,981	0.289	0.196	0.132	0.233
Trinity	21,537	5,713	15,824	11,143	0.339	0.265	0.175	0.298
SOAPdenovo-trans	26,257	3,577	22,680	13,279	0.212	0.136	0.090	0.166
rnaSPAdes	21,469	5,317	16,152	11,539	0.315	0.248	0.161	0.277
Concatenation	76,496	5,001	71,495	11,855	0.297	0.065	0.057	0.107
EvidentialGene	73,539	4,876	68,663	11,980	0.289	0.066	0.057	0.108
ConSemble	16,401	6,496	9,905	10,360	0.385	0.396	0.243	0.391
Minsemble	16,853	7,811	9,042	9,045	0.463	0.463	0.302	0.463

^a Total number of benchmark genes is 16,856.^b Total number of clusters generated by MinIsoClust.Table A.15: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *A. thaliana* Col0 dataset for the single-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	16,624	5,361	4,141	11,263	0.564	0.322	0.258	0.410
Trinity	10,866	5,849	3,653	5,017	0.616	0.538	0.403	0.574
SOAPdenovo-trans	17,117	4,261	5,241	12,856	0.448	0.249	0.191	0.320
rnaSPAdes	18,878	5,547	3,955	13,331	0.584	0.294	0.243	0.391
Concatenation	17,892	4,883	4,619	13,009	0.514	0.273	0.217	0.357
EvidentialGene	10,381	5,064	4,438	5,317	0.533	0.488	0.342	0.509
ConSemble	10,433	6,197	3,305	4,236	0.652	0.594	0.451	0.622
Minsemble	11,791	6,361	3,141	5,430	0.669	0.539	0.426	0.597

^a Total number of benchmark single-isoform genes is 9,502.^b Total number of clusters generated by MinIsoClust.Table A.16: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the human dataset for the single-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	18,605	4,180	4,395	14,425	0.487	0.225	0.182	0.308
Trinity	15,474	4,637	3,938	10,837	0.541	0.300	0.239	0.386
SOAPdenovo-trans	29,211	4,159	4,416	25,052	0.485	0.142	0.124	0.220
rnaSPAdes	16,048	4,368	4,207	11,680	0.509	0.272	0.216	0.355
Concatenation	16,600	3,320	5,255	13,280	0.387	0.200	0.152	0.264
EvidentialGene	17,771	3,680	4,895	14,091	0.429	0.207	0.162	0.279
ConSemble	13,988	4,691	3,884	9,297	0.547	0.335	0.262	0.416
Minsemble	15,762	5,530	3,045	10,232	0.645	0.351	0.294	0.454

^a Total number of benchmark single-isoform genes is 8,575.^b Total number of clusters generated by MinIsoClust.

Table A.17: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Rice dataset for the single-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	12,268	1,123	11,145	9,713	0.104	0.092	0.051	0.097
Trinity	8,938	1,298	7,640	9,538	0.120	0.145	0.070	0.131
SOAPdenovo-trans	17,306	3,078	14,228	7,758	0.284	0.178	0.123	0.219
rnaSPAdes	8,750	2,668	6,082	8,168	0.246	0.305	0.158	0.272
Concatenation	11,740	4,085	7,655	6,751	0.377	0.348	0.221	0.362
EvidentialGene	7,997	2,113	5,884	8,723	0.195	0.264	0.126	0.224
ConSemble	7,931	4,996	2,935	5,840	0.461	0.630	0.363	0.532
Minsemble	7,459	5,277	2,182	5,559	0.487	0.707	0.405	0.577

^a Total number of benchmark single-isoform genes is 10,836.

^b Total number of clusters generated by MinIsoClust.

Table A.18: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Soybean dataset for the single-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	13,782	3,128	10,654	13,207	0.191	0.227	0.116	0.208
Trinity	14,312	2,291	12,021	14,044	0.140	0.160	0.081	0.150
SOAPdenovo-trans	9,986	5,501	4,485	10,834	0.337	0.551	0.264	0.418
rnaSPAdes	12,871	4,883	7,988	11,452	0.299	0.379	0.201	0.334
Concatenation	9,778	5,934	3,844	10,401	0.363	0.607	0.294	0.454
EvidentialGene	14,561	4,731	9,830	11,604	0.290	0.325	0.181	0.306
ConSemble	11,394	8,206	3,188	8,129	0.502	0.720	0.420	0.592
Minsemble	13,763	8,898	4,865	7,437	0.545	0.647	0.420	0.591

^a Total number of benchmark single-isoform genes is 16,335.

^b Total number of clusters generated by MinIsoClust.

Table A.19: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *Z. mays* B73 dataset for the single-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	13,826	982	12,844	7,473	0.116	0.071	0.046	0.088
Trinity	14,452	887	13,565	7,568	0.105	0.061	0.040	0.077
SOAPdenovo-trans	9,210	673	8,537	7,782	0.080	0.073	0.040	0.076
rnaSPAdes	13,981	1,269	12,712	7,186	0.150	0.091	0.060	0.113
Concatenation	17,862	2,650	15,212	5,805	0.313	0.148	0.112	0.201
EvidentialGene	23,383	1,412	21,971	7,043	0.167	0.060	0.046	0.089
ConSemble	14,274	3,487	10,787	4,968	0.412	0.244	0.181	0.307
Minsemble	13,121	3,869	9,252	4,586	0.458	0.295	0.219	0.359

^a Total number of benchmark single-isoform genes is 8,455.

^b Total number of clusters generated by MinIsoClust.

Table A.20: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *Z. mays* Mo17 dataset for the single-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	22,210	4,693	17,517	11,642	0.287	0.211	0.139	0.244
Trinity	18,854	5,383	13,471	10,952	0.330	0.286	0.181	0.306
SOAPdenovo-trans	25,358	3,406	21,952	12,929	0.209	0.134	0.089	0.163
rnaSPAdes	18,521	5,032	13,489	11,303	0.308	0.272	0.169	0.289
Concatenation	65,000	4,696	60,304	11,639	0.287	0.072	0.061	0.115
EvidentialGene	65,896	4,589	61,307	11,746	0.281	0.070	0.059	0.112
ConSemble	14,394	6,111	8,283	10,224	0.374	0.425	0.248	0.398
Minsemble	15,682	7,366	8,316	8,969	0.451	0.470	0.299	0.460

^a Total number of benchmark single-isoform genes is 16,335.^b Total number of clusters generated by MinIsoClust.Table A.21: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *A. thaliana* Col0 dataset for the multiple-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	1,610	9	2,512	1,601	0.004	0.006	0.002	0.004
Trinity	3,170	221	2,300	2,949	0.088	0.070	0.040	0.078
SOAPdenovo-trans	1,899	156	2,365	1,743	0.062	0.082	0.037	0.071
rnaSPAdes	5,127	155	2,366	4,972	0.061	0.030	0.021	0.041
Concatenation	10,303	677	1,844	9,626	0.269	0.066	0.056	0.106
EvidentialGene	13,445	291	2,230	13,154	0.115	0.022	0.019	0.036
ConSemble	2,913	599	1,922	2,314	0.238	0.206	0.124	0.220
Minsemble	4,667	1,005	1,516	3,662	0.399	0.215	0.163	0.280

^a Total number of benchmark multiple-isoform genes is 2,521.^b Total number of clusters generated by MinIsoClust.Table A.22: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Human dataset for the multiple-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	1,565	156	3,427	1,409	0.044	0.100	0.031	0.061
Trinity	2,211	401	3,182	1,810	0.112	0.181	0.074	0.138
SOAPdenovo-trans	1,328	152	3,431	1,176	0.042	0.114	0.032	0.062
rnaSPAdes	1,945	315	3,268	1,630	0.088	0.162	0.060	0.114
Concatenation	3,076	254	3,329	2,822	0.071	0.083	0.040	0.076
EvidentialGene	9,655	242	3,341	9,413	0.068	0.025	0.019	0.037
ConSemble	2,236	523	3,060	1,713	0.146	0.234	0.099	0.180
Minsemble	6,480	676	2,907	5,804	0.189	0.104	0.072	0.134

^a Total number of benchmark multiple-isoform genes is 3,583.^b Total number of clusters generated by MinIsoClust.

Table A.23: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Rice dataset for the multiple-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	397	5	392	219	0.022	0.013	0.008	0.016
Trinity	758	1	757	223	0.004	0.001	0.001	0.002
SOAPdenovo-trans	331	1	330	223	0.004	0.003	0.002	0.004
rnaSPAdes	1,830	3	1,827	221	0.013	0.002	0.001	0.003
Concatenation	2,787	6	2,781	218	0.027	0.002	0.002	0.004
EvidentialGene	11,172	7	11,165	217	0.031	0.001	0.001	0.001
ConSemble	3,474	18	3,456	206	0.080	0.005	0.005	0.010
Minsemble	4,905	44	4,861	180	0.196	0.009	0.009	0.017

^a Total number of benchmark multiple-isoform genes is 224.^b Total number of clusters generated by MinIsoClust.Table A.24: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the Soybean dataset for the multiple-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	1,860	101	1,759	551	0.155	0.054	0.042	0.080
Trinity	1,120	87	1,033	565	0.133	0.078	0.052	0.098
SOAPdenovo-trans	1,922	119	1,803	533	0.183	0.062	0.048	0.092
rnaSPAdes	1,918	96	1,822	556	0.147	0.050	0.039	0.075
Concatenation	9,281	113	9,168	539	0.173	0.012	0.012	0.023
EvidentialGene	4,686	69	4,617	583	0.106	0.015	0.013	0.026
ConSemble	2,343	157	2,186	495	0.241	0.067	0.055	0.105
Minsemble	1,860	354	1,506	298	0.543	0.190	0.164	0.282

^a Total number of benchmark multiple-isoform genes is 652.^b Total number of clusters generated by MinIsoClust.Table A.25: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *Z. mays* B73 dataset for the multiple-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	1,816	46	1,770	2,912	0.016	0.025	0.010	0.019
Trinity	1,835	39	1,796	2,919	0.013	0.021	0.008	0.016
SOAPdenovo-trans	1,888	29	1,859	2,929	0.010	0.015	0.006	0.012
rnaSPAdes	3,014	77	2,937	2,881	0.026	0.026	0.013	0.026
Concatenation	10,274	81	10,193	2,877	0.027	0.008	0.006	0.012
EvidentialGene	13,382	60	13,322	2,898	0.020	0.004	0.004	0.007
ConSemble	1,729	78	1,651	2,880	0.026	0.045	0.017	0.033
Minsemble	9,006	277	8,729	2,681	0.094	0.031	0.024	0.046

^a Total number of benchmark multiple-isoform genes is 2,958.^b Total number of clusters generated by MinIsoClust.

Table A.26: Comparison of transcriptome assembly performance among *de novo* and ensemble methods for the *Z. mays* Mo17 dataset for the multiple-isoform genes^a

Assembler	Total ^b	TP	FN	FP	Recall	Precision	Accuracy*	F-measure
IDBA-Tran	2,706	4	2,702	517	0.008	0.001	0.001	0.002
Trinity	2,683	117	2,566	404	0.225	0.044	0.038	0.073
SOAPdenovo-trans	899	2	897	519	0.004	0.002	0.001	0.003
rnaSPAdes	2,948	47	2,901	474	0.090	0.016	0.014	0.027
Concatenation	11,496	66	11,430	455	0.127	0.006	0.006	0.011
EvidentialGene	7,643	47	7,596	474	0.090	0.006	0.006	0.012
ConSemble	2,007	119	1,888	402	0.228	0.059	0.049	0.094
Minsemble	2,334	244	2,090	277	0.468	0.105	0.093	0.171

^a Total number of benchmark multiple-isoform genes is 521.

^b Total number of clusters generated by MinIsoClust.

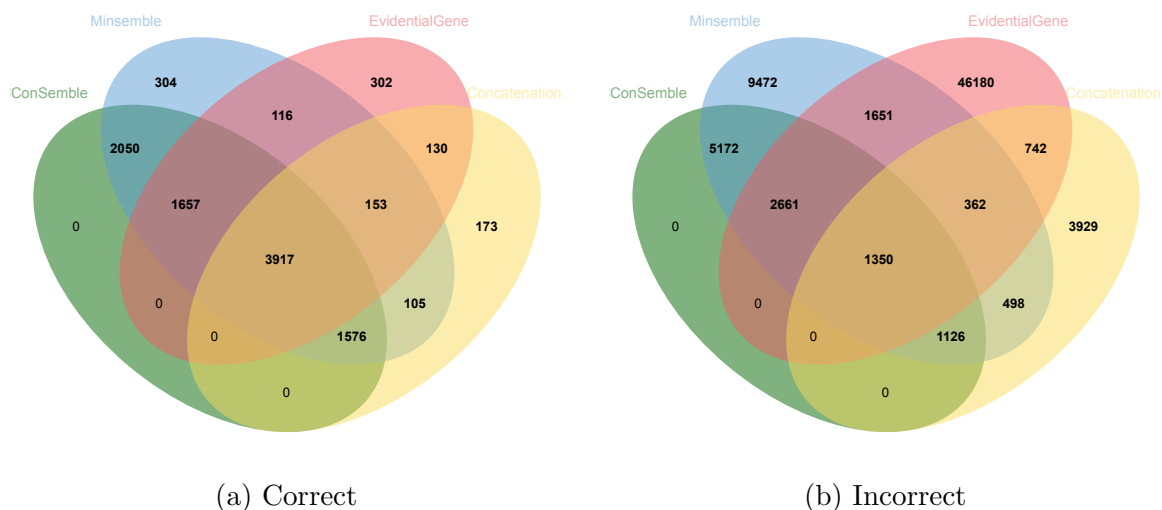


Figure A.1: Numbers of correctly and incorrectly assembled contigs for the Human benchmark dataset shared among the four ensemble assembly approaches.