UNIVERSITY OF CALIFORNIA

Los Angeles

Computational Methods for the Analysis of Genomic and Proteomic Sequences

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Jui-Ting Ju

2019

ABSTRACT OF THE DISSERTATION

Computational Methods for the Analysis of
Genomic and Proteomic Sequences

by

Jui-Ting Ju
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2019
Professor Wei Wang, Chair

The rapid generation of biological sequences, such as nucleotide and amino acid sequences, has revolutionized the studies in the field of molecular biology. To name a few applications, DNA sequences generated by the RNA-Sequencing technology facilitate the studies of gene expression analysis; protein sequences represent the primary structure to predict protein-protein interactions. Moreover, the vast amount of sequence data generated from high-throughput technologies gears up the data analysis to the omics level. As a consequence, developing novel computational methods and tailoring existing algorithms are highly imperative to extract relevant and critical knowledge from sequence data.

In this dissertation, we introduce several computational frameworks that leverage the genomic sequences to quantify gene expression and utilize the proteomic sequences to characterize protein-protein interactions. The methodologies presented in these frameworks span different research areas, including feature extraction from string data, string matching for DNA sequence, statistical inference for expression quantification, and sequence-pair modeling through deep learning. As a result, these approaches not only tackle specific challenges in the applications mentioned above but also present the potentials to address issues in other sequencing applications.

The dissertation of Jui-Ting Ju is approved.

University of California, Los Angeles

2019

To my parents,

for their endless love and supports;

my late grandparents,

for their beliefs in the power of education.

TABLE OF CONTENTS

List of Figures . . . . **ix**

List of Tables . . . . **xi**

Acknowledgments . . . . **xii**

Vita . . . . **xiv**

1 Introduction **1**

1.1 Scope of the Research 2

1.2 Contributions 3

1.3 Overview 4

2 Related Work **5**

2.1 Alignment-Based Approaches for Expression Quantification 5

2.2 Quantification of Pseudogenes 6

2.3 Lightweight Approaches for Expression Quantification 6

2.4 K-mers Counters 7

2.5 Protein-Protein Interaction Predictions 8

3 Correcting Read Alignment for Pseudogenes **9**

3.1 Model 9

3.2 Homologous Community Partition 12

3.3 Read Count Profile Classification 13

3.4 Training Stage 13

3.5 Validation Stage 14

3.6 Read Re-assignment 15

4 Quantification of RNA-Seq via Variable-Length *k*-mers 16

4.1 *Sig-mers* Identification 17

4.2 *Sig-mers* Selection 19

4.3 RNA-Seq Reads Matching 21

4.3.1 Aho-Corasick Automaton 23

4.3.2 Thinned Automaton with Binarized Pattern Matching 23

4.3.3 Acceleration by Rolling Hash 25

4.4 Transcript Abundance Estimation 27

5 Multifaceted Protein-Protein Interaction Prediction Based on Siamese Residual RCNN 30

5.1 RCNN-based Protein Sequence Encoder 31

5.1.1 Residual RCNN 32

5.1.2 Protein Sequence Encoding 34

5.1.3 Pre-trained Amino Acid Embeddings 35

5.2 Learning Architecture and Learning Objectives 36

5.2.1 Siamese Architecture 36

5.2.2 Learning Objectives 36

6 Datasets 37

6.1 NGS Datasets 37

6.1.1 Psuedogene Reference 37

6.1.2 Reference Transcriptome 37

6.1.3 Signatures Used in TahcoRoll 38
6.1.4 Simulated Reads for Pseudogene 38
6.1.5 Simulated Reads for Fleximer and TahcoRoll 38
6.1.6 Real Data from Human BodyMap 39
6.1.7 Real Data from Different Sequencing Platforms 39
6.2 Protein-Protein Interaction Datasets 40
6.2.1 Guo's Datasets 40
6.2.2 STRING Datasets. 40
6.2.3 SKEMPI Dataset. 41

7 Experiments and Results 42
7.1 Alignment Correction for Pseudogene Abundance Estimates 42
7.1.1 Misalignment of Pseudogene and Its Homologous Parent 42
7.1.2 Community Detection for Homologous Genes 44
7.1.3 k-NN Classification for Homologous Community 46
7.1.4 Read Count Estimation for Homologous Community 47
7.2 Transcript Quantification via Variable Length k-mers 48
7.2.1 Motivating Examples 49
7.2.2 *k-mers* Generation 50
7.2.3 Simulation Study 51
7.2.4 Human BodyMap 2.0 Project 52
7.3 Signature Profiling via Thinned Aho-Corasick Automaton 54
7.3.1 Experimental Settings 55
7.3.2 Automaton Construction 55
7.3.3 Pilot Study of 13 Approaches 56

7.3.4 Extensive Study on Synthetic Datasets 58

7.3.5 Real Datasets from Different Sequencing Platforms 60

7.4 Characterizing Protein-Protein Interaction via Deep Learning 61

7.4.1 Binary PPI Prediction 62

7.4.2 Interaction Type Prediction 65

7.4.3 Binding Affinity Estimation 67

7.4.4 Amino Acid Embeddings 69

7.4.5 Run-time Analysis 69

8 Conclusion 71

A Background on Suffix Tree 74

B Propositions in TahcoRoll 76

B.1 Proof of Proposition 1 76

B.2 Proof of Proposition 2 77

B.3 Proof of Proposition 3 77

C Hyperparameter Study for PIPR 79

D Software Configuration for *K-mer* Counters 81

D.1 Thread-Safe Shared Memory Hashing 81

D.2 Disk-Based Hashing 81

D.3 Probabilistic Hashing 82

D.4 Suffix-Arrays 83

D.5 Burst Tries 83

LIST OF FIGURES

3.1 The overall framework of correcting read alignment for pseudogenes. 10

4.1 An overview of Fleximer. 17
4.2 *Sig-mers* identification in Fleximer. 19
4.3 *Sig-mers* selection and matching in Fleximer 21
4.4 The raw and binarize Aho-Corasick automaton with five signatures. 26
4.5 Collisions in binarized representations. 26

5.1 The overall learning architecture of PIPR. 31
5.2 The structure of the residual RCNN encoder in PIPR. 31

7.1 Alignment profile between ENST00000421244 and its pseudogene. 43
7.2 Relationship between expected and observed read counts. 43
7.3 Homologous communities partition and size distribution. 45
7.4 Run time and accuracy of different community detection algorithms. 46
7.5 Parameter selection for kNN-classification. 47
7.6 Evaluation of the TPM estimation over different ks. 49
7.7 *K-mer* statistics of different methods. 50
7.8 Accuracy and efficiency evaluations using simulated data. 51
7.9 Read recovery rate of different methods. 52
7.10 Evaluation of Human BodyMap with Venn diagrams. 53
7.11 Evaluation of Human BodyMap with accuracy metrics. 54
7.12 Run-time and Memory for automaton construction. 56
7.13 Run-time and memory for pilot study. 57

7.14 Run-time for parallel study. 60

7.15 Mutation effects on structure and binding affinity. 68

C.1 Hyperparameter Study for PIPR. 80

LIST OF TABLES

7.1 Description of Simulated Datasets. 44

7.2 Overall prediction errors. 48

7.3 Evaluation of read count estimations. 48

7.4 Run-time and memory of different read sets. 58

7.5 Run-time and memory for profiling different signature sets. 59

7.6 Run-time and memory of real datasets across different sequencing platforms. . . 61

7.7 Evaluation of Different Binarized Representations. 61

7.8 Evaluation of binary PPI prediction on the Yeast dataset. 64

7.9 Statistical assessment on the accuracy of binary PPI prediction. 64

7.10 Evaluation of binary PPI prediction on variants of multi-species dataset. 65

7.11 Accuracy (%) and fold changes over zero rule for PPI interaction type prediction. 66

7.12 Evaluation of binding affinity prediction on the SKEMPI dataset. 68

7.13 Comparison of amino acid representations based on binary prediction. 69

7.14 Run-Time of training embeddings and different prediction tasks. 70

ACKNOWLEDGMENTS

The accomplishment of my Ph.D. journey was not possible without the support and guidance of my supervisor, Professor Wei Wang. I would like to express my highest appreciation to her for her patience and dedication. Most importantly, for offering me the opportunity to pursue a Ph.D. in computational biology. Throughout the past six years, she has taught me not only scientific knowledge but also skill-sets to make myself a more competitive professional. To name a few, a critical mindset to assess scientific works, a calm and bright attitude to navigate through crisis and frustration, and the values of prioritization to increase research productivity. Although we may disagree with each other, she has always been patient and supportive by the end of the day. With all of these, I am indebted to her.

I also own my sincere gratitude to all my thesis committee members. Professor Eleazar Eskin has encouraged me and supported my decision to pursue a Ph.D. in computational biology. His computational genetic class has inspired me to continue exploring the challenges and excitements in this field. In his class, he has taught me to formulate the challenging biological questions into attainable computational problems. Professor Stott Parker has been a great mentor since I was pursuing my Master's study at UCLA. I have always enjoyed our one-to-one conversation with his fresh perspective and valuable advice. Professor Carlo Zaniolo and I have collaborated on a few projects together. I truly appreciate his help and support throughout our collaborations. Specifically, he always initiates many warm and kind conversations with the students in ScAI Lab. I am also grateful to have Professor Jessica Li serving as one of my committee members. Even though we did not have a chance to collaborate, she has always made herself available whenever I have questions on my research.

I have many chances to collaborate with different people at UCLA. Among these collaborations, I would like to express a special thanks to Professor Peipei Ping. She has offered me the opportunities to present and to participate in several NIH meetings. She has also been very generous and supportive of me in attending different conferences. People in her group, including Vincent Kyi, Brian Bleakley, Howard Choi, David Liem, and Sarah Scruggs, have not only taught me the knowledge of cardiovascular diseases but also helped me on attaining many collaborative projects. Other collaborators include Muhao Chen and Xuelu Chen from Professor Carlo Zaniolo's lab. Together, we have discovered the promising potential of deep learning frameworks on different biological applications. The discussions with Dat Duong, Lisa Gai, Jennifer Zhou, and Tevfik Dincer from Professors Eleazar Eskin's and Jason Ernst's labs have also provided me a different scientific mindset.

Besides the interdisciplinary collaborations, I would like to thank each of the members of Professor Wang's lab. Ruirui Li and I joined the lab around the same time, and we witnessed all the events in ScAI lab together. I appreciate his support and presence on all the ups and downs throughout the years. The joining of Wenchao Yu, Yichao Zhou (Joey), Jyun-Yu Jiang, Zeyu Li, Nathan LaPierre, Guangyu Zhou, Jungheng Hao (Jeff), and Xiusi Chen have made the lab much more collaborative and interactive. We have created a positive research environment where we discuss and challenge each other. I am very grateful to have them as my second family in Los Angeles.

I would also like to offer a heartfelt thanks to my close friends at UCLA: Farhad Hormozdiari, Petko Fiziev, Larry Lam, Ai Sasho, and Robert Brown. We studied for classes together, brainstormed on various bioinformatics projects, discussed a diverse range of research topics, and explored different cuisine and places around the city. We also shared many important moments and life events throughout the years. Our friendship is the greatest treasure supporting me through both the good and bad times.

Last but not least, to the most important people in my life, my parents (Amy Liao and Nai-Chang Ju) and my sister (Chloe Ju). I am forever indebted to them for their faith and endless love. They may not necessarily understand the challenges of this journey, but they have always given me the freedom and courage to pursue anything that I set my mind to.

VITA

2001 – 2006	B.Sc. (Biological Science), University of Alberta, Edmonton, Canada
2006 – 2009	Research Technologist, University of Alberta, Edmonton, Canada
2009 – 2011	Bioinformatician, Qteros, Inc., Marlborough, Massachusetts
2012 – 2012	SQA Engineering Intern, Symantec, Inc., Culver City, California
2011 – 2013	M.S. (Computer Science), UCLA.
2014 – 2014	Technical Research Intern, eBay, Inc., San Jose, California
2014 – 2015	Teaching Assistant, Computer Science Department, UCLA
2016 – 2016	Bioinformatics Intern, Natera, Inc., San Carlos, California
2013 – 2019	Graduate Student Researcher, Computer Science Department, UCLA

PUBLICATIONS

Parts of the work in this thesis have appeared in the following publications:

Muhao Chen*, **Chelsea J.-T. Ju***, Guangyu Zhou, Tianran Zhang, Xuelu Chen, Kai-Wei Chang, Carlo Zaniolo and Wei Wang. Multifaceted protein-protein interaction prediction based on Siamese residual RCNN. *Bioinformatics* (ISMB/ECCB 2019).

Chelsea J.-T. Ju*, Jyun-Yu Jiang*, Ruirui Li, Zeyu Li and Wei Wang. TahcoRoll: An efficient approach for signature profiling in genomic Data through variable-length k-mers. *bioRxiv*, 2017. pre-print.

Chelsea J.-T. Ju, Ruirui Li, Zhengliang Wu, Jyun-Yu Jiang, Zhao Yang and Wei Wang. Fleximer: Accurate quantification of RNA-Seq via variable-Length k-mers. In *Proceedings of The 8th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics* (ACMBCB 2017).

Chelsea J.-T. Ju, Zhuangtian Zhao and Wei Wang. Efficient approach to correct read alignment for pseudogene abundance estimates. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*,14(3): 522-533, 2017.

Chelsea J.-T. Ju, Zhuangtian Zhao and Wei Wang. PseudoLasso: leveraging read alignment in homologous regions to correct pseudogene expression estimates via RNASeq. In *Proceedings of The 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics* (ACMBCB 2014).

Other relevant publications:

Nathan LaPierre, **Chelsea J.-T. Ju**, Guangyu Zhou and Wei Wang. MetaPheno: A critical evaluation of deep learning and machine learning in metagenome-based disease prediction. *Methods*, 2019.

Ruirui Li, Jyun-Yu Jiang, **Chelsea J.-T. Ju** and Wei Wang. CORALS: Who are my potential new customers? Tapping into the wisdom of customers' decisions. In *Proceedings of The 12th ACM International Conference on Web Search and Data Mining* (WSDM 2019).

Guangyu Zhou, Jyun-Yu Jiang, **Chelsea J.-T. Ju** and Wei Wang. Inferring microbial communities for city scale metagenomics using neural networks. In *Proceedings of 2018 IEEE International Conference on Bioinformatics and Biomedicine* (BIBM 2018).

CHAPTER 1

Introduction

The advances of high throughput technologies have an incredible impact in the field of molecular biology, facilitating the movement of conventional experiments to the "-omics" research. In genomics, high throughput sequencing (HTS) is the leading technology to study genome-wide analysis. One of the applications, RNA-Sequencing (RNA-Seq), captures the snapshot of existing RNA for gene expression quantification by reading out the transcript sequences into millions of short reads. Following a series of data analysis, the number of reads from a transcript can be used to estimate the expression abundance of that transcript. In proteomics, mass spectrometry has been the predominant method for *de novo* peptide sequencing, providing the primary structure for protein complexes. Leveraging the protein sequences, many downstream data analyses can be carried out, including but are not limited to, functional annotation, predicting protein-protein interaction, and characterizing the interaction types. Regardless of the sequence type (i.e. the nucleic acid sequence for DNA or the amino acid sequence for protein), knowledge extracted from the sequence information allows researchers to answer certain biological questions of interest. As a result, the development of computational tools and algorithms is critical to cope with the rapid generation of sequencing data, and to facilitate a wide range of tasks in data analysis. In this dissertation, we study three different research problems in analyzing the biological sequences. The first two problems focus on leveraging the RNA-Seq reads to quantify the expression of gene isoform. The third problem focuses on analyzing a pair of protein sequences to decipher their interaction properties. For each problem, we discuss the limitations of existing approaches, and propose new methods to address these challenges.

1.1 Scope of the Research

The advent of RNA-Seq poses substantial computational problems, specifically in handling the massive amount of read data [98]. Two of these problems are summarized below. The amino acid sequence serves as the primary structure of a protein complex, which is the simplest type of information obtained through direct sequencing or translated from DNA sequences. For the third research problem, we discuss the challenges and potential of characterizing a pair of proteins directly from the sequence information.

- **Read alignment in low complexity regions.** The data analysis of an RNA-Seq experiment starts from aligning short reads to the reference genome or transcriptome. However, existing aligners lack the sensitivity to distinguish reads that come from homologous regions of a genome. One group of these homologies is the paralog pseudogenes. Pseudogenes arise from duplication of a set of protein-coding genes, and share a significant amount of sequence similarity with their parent genes. They have been considered as degraded paralogs in the genome due to their loss of functionality. Recent studies have provided evidence to support their novel regulatory roles in biological processes. With the growing interests in quantifying the expression level of pseudogenes, it is critical to have a sensitive method that can correctly align ambiguous reads and accurately estimate the expression level among homologous genes.

- **Quantification of RNA-Seq via variable length *k*-mers.** The alignment step of the data analysis requires a significant amount of computational resources [23]. Even with parallel computing, analyzing a reasonable size of RNA-Seq experiment can still take hours [70]. To alleviate this computational bottleneck, a series of methods have been proposed to perform a lightweight quantification in an alignment-free manner. These methods utilize the notion of k-mers, which are short consecutive sequences representing the signatures of each transcript, to estimate the relative abundance from RNA-Seq reads. Existing k-mers based approaches make use of a set of fixed size k-mers; however, selecting the appropriate k is not intuitive. Additionally, the best k

to characterize a transcript sequence varies for different transcripts. Thus, an efficient approach to identify and select an optimal set of variable length k-mers is needed.

- **Characterization of protein-protein interactions via protein sequences.** Protein-protein interaction (PPI) prediction represents a fundamental computational biology problem. Existing efforts focus on extracting predefined features from individual sequences, followed by applying statistical algorithms to classify the PPIs. These explicit features are dedicated to specific facets of the protein profiles, and hence provide limited coverage on the PPI information. Evidently, an efficient mechanism is needed to apprehend the mutual influence of protein pairs. It is also essential to have a framework that can be generalized to different prediction tasks, such as binary PPI prediction, interaction type prediction, and binding affinity estimation.

1.2 Contributions

In this dissertation, we emphasize the importance of the research problems mentioned above, and identify the specific challenges fall within each research problem. We propose a series of computational methods to tackle these challenges.

The first issue of analyzing the RNA-Seq data lies on aligning reads that come from regions with low sequence complexity. Reads from these regions can be aligned to more than one position in the genome. Incorrect alignment leads to inaccurate expression abundance estimation, which poses the major challenge for quantifying the expression of homologous genes. We propose a linear regression approach to learn the alignment behaviors among homologous genes. Since the homologous genes are sparse throughout the genome, we first incorporate a community detection algorithm to cluster genes into different groups based on sequence similarity. Following the partition, read count estimation can be efficiently and concurrently performed on smaller sets of genes through the linear regression model. Once the read count of each gene is accurately estimated, reads are re-assigned to regions in which the observed read count is lower than the estimation. Results show that our approach is able to accurately estimate abundance and correctly assign reads among homologous genes.

The second research problem addresses the caveats of using the fixed length of k-mers to analyze RNA-Seq data. We propose a novel framework, Fleximer, containing four modules to estimate the transcript abundance using k-mers with different lengths. Fleximer takes a predefined genomic or transcriptomic partition as an input. The partition can be based on sequence similarity, biological functions, or any other user-defined scheme. To discover k-mers that are unique to each cluster with all possible lengths, we rely on the structure and properties of a suffix tree [8]. Fleximer also provides a feasible data structure that builds upon the Aho-Corasick algorithm [1] to store and to profile these k-mers in sequencing data efficiently. Experimental results have shown that the selected k-mers own more distinguishing features, and thus substantially reduce the errors in transcript abundance estimation.

The third work focuses on modeling a pair of protein sequences to predict their interactions. We present an end-to-end framework, PIPR, which incorporates a deep residual recurrent convolutional neural network in the Siamese architecture. This architecture leverages both robust local features and contextualized information, which are significant for capturing the mutual influence of a pair of protein sequences. Experimental evaluations show that PIPR outperforms various state-of-the-art systems on the binary PPI prediction. It also shows a promising performance on more challenging problems of interaction type prediction and binding affinity estimation.

1.3 Overview

The rest of the dissertation is organized as follows: Chapter 2 summarizes the relevant works for each research problem. Chapters 3 and 4 describe our methods in addressing the challenges of transcript quantification via RNA-Seq. Chapter 5 presents our framework in studying the protein sequence pairs. Chapter 6 provides the description of the datasets used in different experiments. Chapter 7 presents the experiments and findings. Chapter 8 concludes this dissertation with a summary of the works and the plan of future extensions.

CHAPTER 2

Related Work

2.1 Alignment-Based Approaches for Expression Quantification

The conventional approaches for the RNA-Seq pipeline can be divided into three stages [23]: read mapping, transcriptome reconstruction, and expression quantification. Millions of short reads are first aligned either to a reference transcriptome or genome. Several commonly used aligners include Tophat [38], MapSplice [95], and SpliceMap [4]. In transcriptome reconstruction, overlapped alignments from the first stage are aggregated and assembled into transcripts. The abundance of each transcript is quantified based on the number of mapped reads, and normalized to account for transcript size and the total number of mapped reads. Cufflinks [91], RSEM [44], and eXpress [76] are the leading approaches that employ an expectation-maximization (EM) algorithm [66] to iteratively reconstruct the transcriptome and estimate the transcript abundances. Statistical analysis can be further applied to identify significant changes in gene expression across different experiments.

During the course of data analysis, each step has a cascading effect, and the outcome from read alignment can predominantly change the results of any downstream analysis. Most Eukaryotic genomes, including the human genome, are full of large repeated segments. As a result, reads generated from these regions can be mapped to more than one place in the genome. This type of reads is referred to as the “multiread”. Existing aligners either mis-align these reads [47] or employ a scoring system to keep the “best” alignment [111]. However, a read can still have multiple alignments with equally good scores. In a situation like this, it remains difficult to identify the true origin of a read.

2.2 Quantification of Pseudogenes

One type of the repeat sequences in the genome is the pseudogenes, which has received notable attention due to the novel discovery of their regulatory roles in different biological processes [28, 50, 114]. In the effort of quantifying the expression of pseudogene, Tonner et al. [90] developed a pipeline that created a composite genome to include the human genome and the mRNA sequence of ribosomal protein genes. Their method keeps only the uniquely mapped read for abundance estimation. The idea of discarding all multireads is one of the most common practices in resolving the multiple alignment issues. It is easy to apply, but tosses out critical information for the quantification step. As a result, it creates a bias toward genes or pseudogenes with more unique sequences in the genome. Another approach is to assign the multireads based on the mapping quality. Nevertheless, there can be more than one alignment that shares the same quality score due to sequence similarity. Other approaches include assigning multireads to the best locus based on local coverage estimation from uniquely mapped reads [62], or based on probabilistic models [44, 69]. However, none of these approaches leverages the relationship of read alignment among homologous regions.

2.3 Lightweight Approaches for Expression Quantification

In the recent development of RNA-Seq data analysis, alignment-free methods have been proposed to alleviate the computational burden of read alignment. Sailfish [70] is the first implementation to use k-mers for transcript quantification. It indexes all k-mers that appear at least once in the transcriptome and counts their occurrences in RNA-Seq data. Based on these counts, it estimates the relative transcript abundance through the EM algorithm. Further ameliorating the throughput, RNA-Skim [110] divides the transcriptome into clusters based on sequence similarity. For each cluster, it selects a set of k-mers that do not appear in any other clusters, denoting as *sig-mers* (signatures of a cluster). Since each cluster contains a unique set of k-mers, the quantification step can be performed independently for smaller groups of transcripts. Kallisto [9] builds targeted de-Bruijn graphs on all k-mers to facilitate the quantification, and claims faster and more accurate performance than Sailfish. All these approaches have demonstrated a promising performance, improving the running time from hours to minutes compared to the alignment-based approaches.

Selecting the best k can be challenging for a new experiment. A k-mer cannot be too short because it can randomly match to a read that comes from a different transcript than the k-mer origins. On the other hand, a longer k-mer is less robust to read containing sequencing errors or individual variations. Sailfish originally set the default value of k to be 20, but changed it to 31 according to its website. The initial choice is based on the evaluations over a small range of k (15 - 25) without further optimization. RNA-Skim studies the overall accuracy of transcript abundances for different k before setting it to 60 as the default option. Neither of them has provided any guidance in selecting the right k for a new experiment.

2.4 *K*-mers Counters

The success of the lightweight approach in processing the sequencing data is fostered by the rapid development of efficient algorithms in k-mer counting. Existing k-mer counters index reads into a compact and searchable structure, such as a hash table, a burst trie, or a compact suffix array. The occurrences of a specific k-mer can be retrieved by querying these data structures. Jellyfish [54] has been widely used as the underlying structure for Sailfish, Kraken [100] which assigns taxonomic labels for metagenomic reads, and DIAMUND [82] which detects mutation without whole-genome alignment. It exploits the compare-and-swap assembly instruction to update a memory location in a multi-threaded environment. With a similar approach, Squeakr [68] employs the thread-safe approach to efficiently query the counts of a specific k-mer. Probabilistic hashing is also commonly used in k-mer counting. Its implementations include BFCounter [58] and khmer [108]. Disk-based hashing is another popular technique, and its related algorithms are DSK [75], MSPkmerCounter [49], and KMC [17, 18, 41]. Other data structures include burst tries used in KCMBT [53], and suffix-array structures employed in Tallymer [43] and MSBWT [32].

Several implementations, such as khmer and KCMBT, restrict the choice of k to fall in a threshold to mitigate the memory consumption and running time. Suffix-array based approach is the only one that presents the potential to process k-mers of variable lengths. Other methods are designed to process sequences with a fixed k. Thus, repeating the process for different ks is unavoidable, which limits the analysis on a small range of k's.

2.5 Protein-Protein Interaction Predictions

To predict the binary form of PPIs, homology-based methods [72] rely on BLAST [3] to map a pair of sequences to known interacting proteins. Alternatively, other works address the task with statistical learning models, including SVM [25, 106], kNN [101], Random Forest [99], multi-layer perceptron (MLP) [20], and ensemble ELM (EELM) [105]. These approaches rely on feature extraction of the protein sequences, such as conjoint triads (CT) [87, 105], autocovariance (AC) [25, 87, 105], composition-transition-distribution (CTD) descriptors [20, 101], multi-scale continuous and discontinuous (MCD) descriptors [105], and local phase quantization (LPQ) [99]. These features measure physicochemical properties of the amino acids, and aim at summarizing sequence information relevant to PPIs. More recent works [87, 96] propose stacked autoencoders (SAE) to refine these heterogenous features in low-dimensional spaces. Leveraging the deep learning architectures, DPPI [26] employs a convolutional neural network (CNN) to capture the local features from protein profiles. However, it requires excessive efforts to obtain protein profiles through PSI-BLAST [3]. Most importantly, its architecture does not captures the contextualized and sequential features. DNN-PPI [45] represents another relevant work of this line, which uses two separated CNN encoders. However, DNN-PPI does not incorporate physicochemical properties into amino acid representations, and fails to characterize pairwise relations of sequences.

Fewer efforts have been made towards multi-class prediction to infer the interaction types [85, 113] and the regression task to estimate binding affinity [86, 107]. These methods have largely relied on their capability of extracting and selecting better features, while the extracted features are far from fully exploiting the interaction information.

CHAPTER 3

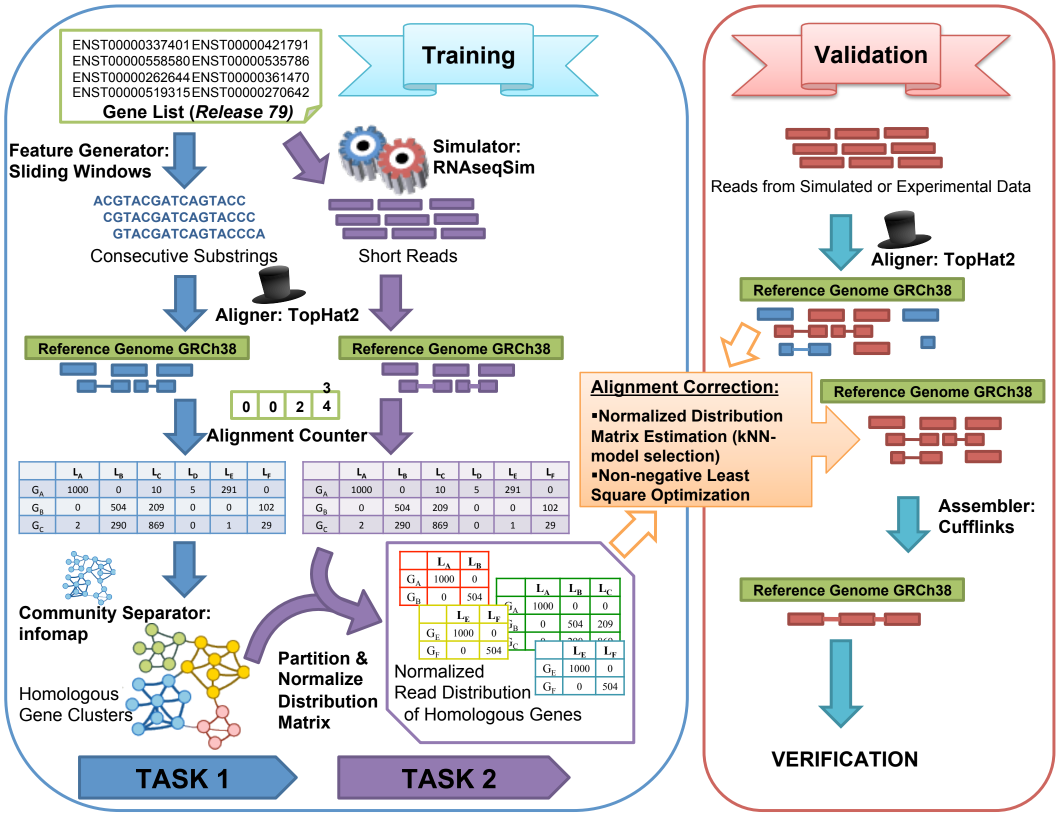
Correcting Read Alignment for Pseudogenes

The ultimate goal of this work is to accurately estimate the read count for each gene, and leverage this information to guide the correction of short read alignments among homologous loci. The overall workflow is illustrated in Figure 3.1. The training stage contains two tasks, which are indicated by different colors: steps with blue arrows describe the first task of feature generation and community detection; steps with purple arrows describe the second task of read distribution computation within each community. In the validation stage, reads from other experiments are first aligned to the reference genome. The alignment profiles are matched to the best normalized matrices from the training stage using k-nearest neighbor classification. The true expressions are estimated using a non-negative least-square model.

3.1 Model

Given a set of read alignment records, the number of reads in each genomic locus can be computed by counting the number of alignments fall into each region. A locus can be a region that spans through a well-annotated gene or pseudogene, or a fragment of an intergenic region. Reads for expressed genes are mostly aligned back to themselves (the corresponding genomic loci), but they can also be misaligned or ambiguously aligned to other homologous loci. Thus, the goal is to estimate the true read counts for these expressed genes. Let s_j be the observed read count for locus i among m loci $(1 < j < m)$, and $\hat{y}_i$ be the estimated read count for gene i among n genes $(1 < i < n)$ [1]. The input of our problem is a vector of $\mathbf{s}$, and the output is a vector of estimated read count $\hat{\mathbf{y}}$.

[1]We use "^" to represent estimated variables



ENST00000337401 ENST00000421791
ENST00000558580 ENST00000535786
ENST00000262644 ENST00000361470
ENST00000519315 ENST00000270642
Gene List (*Release 79*)
Training
Validation
Feature Generator: Sliding Windows
Simulator: RNAseqSim
ACGTACGATCAGTACC
CGTACGATCAGTACCC
GTACGATCAGTACCCA
Consecutive Substrings
Short Reads
Aligner: TopHat2
Reference Genome GRCh38
Alignment Counter
Community Separator: infomap
Homologous Gene Clusters
Partition & Normalize Distribution Matrix
Normalized Read Distribution of Homologous Genes
Alignment Correction:
▪Normalized Distribution Matrix Estimation (kNN-model selection)
▪Non-negative Least Square Optimization
Reads from Simulated or Experimental Data
Assembler: Cufflinks
VERIFICATION
TASK 1
TASK 2

Figure 3.1: The overall framework of correcting read alignment for pseudogenes.

Knowing only the read alignments and read counts for all loci is insufficient to recover the true read counts of expressed genes. However, we can consider reads aligned to each locus come from the corresponding known gene (itself) and its homologous genes. The ratio of this composition is learned from simulated data. We use a distribution matrix $\mathbf{X}$ to keep track of the aligned loci for each gene, where rows represent the origin of reads, and columns represent the destination of the alignment. The distribution matrix is defined below.

Definition 1. *Distribution Matrix. Let* G *be a set of genes,* $\mathrm{G} = \{g_1, g_2, \ldots, g_n\}$*, and* L *be a set of genomic loci with aligned reads,* $\mathrm{L} = \{l_1, l_2, \ldots, l_m\}$*.* $\mathbf{X}$ *is a* $n \times m$ *matrix with* n *genes and* m *loci. Each value in the distribution matrix* x_{ij} *represents the number of reads from gene* g_i *aligned to locus* l_j*.*

Using the distribution matrix, the input of our problem statement is equivalent to the column sum of each locus, i.e. number of aligned reads in a locus. It is mathematically defined in

Equation 3.1, where the errors follow a standard normal distribution.

$$s_j = \sum_{i=1}^{n} x_{ij} + \epsilon, \epsilon \sim N(0,1) \tag{3.1}$$

Assuming that the read distribution among homologous locus follows a linear relation, the general distribution ratio can be learned from simulated replicates. We define a pseudo matrix $\mathbf{A}$ for these ratios, where a_{ij} is the normalized value of x_{ij}.

Definition 2. *Pseudo Matrix. The pseudo matrix $\mathbf{A_{n\times m}}$ is the normalized version of the distribution matrix. It is defined by the proportion of reads mapped to locus l_j out of the total number of reads for a given gene g_i. Let $\mathbf{y}$ be a vector of expected number of reads for genes* G*, $\mathbf{y} = \{y_1, y_2, \ldots, y_n\}$. y_i is defined as the total number of reads simulated from gene g_i. Each value in $\mathbf{A}$ is computed as $a_{ij} = x_{ij}/y_i$.*

With the pseudo matrix, we can rewrite Equation 3.1 as

$$s_j = \sum_{i=1}^{n} y_i a_{ij} + \epsilon, \epsilon \sim N(0,1). \tag{3.2}$$

Putting all n genes and m loci in a matrix form, we have

$$\mathbf{s}_{m\times 1} = \mathbf{A}^T_{m\times n}\mathbf{Y}_{n\times 1} + \epsilon_{m\times 1}, \epsilon \sim N(0,1). \tag{3.3}$$

Reinstating the problem statement using Equation 3.3, $\mathbf{s}$ is the data input, which is a vector of observed read count in each genomic locus, $\mathbf{A}$ is the pseudo matrix obtained from training data, and $\mathbf{y}$ is the unknown read count of expressed genes. The goal is to estimate the read count for all expressed genes with the following objective function,

$$\underset{y_i}{\operatorname{argmin}} \, ||\mathbf{s} - \mathbf{A}^T\mathbf{Y}||^2, \text{ s.t. } y_i \geq 0, \forall y_i \tag{3.4}$$

The number of reads for each gene cannot be negative, and this restriction is reflected in the non-negative constraint in Equation 3.4. In general, there are more loci (m) than expressed genes (n) due to homologous repeats in the genome. The complexity and running time in solving the linear equation increases exponentially with the number of expressed genes. An important observation is that both distribution and pseudo matrices are sparse, since each gene only has a few homologous loci across the genome. The sparsity allows us to partition the matrices into clusters of homologous loci. Each partition can be assessed independently to elevate the computational burden. We use a community detection algorithm to explore and identify these homologous communities.

3.2 Homologous Community Partition

The read count estimation for an expressed gene relies on the information of reads aligned to itself and its homologous loci. A group of homologous loci can be identified by their sequence similarity. From a data mining perspective, a set of objects can be grouped together if they share a certain amount of features. In order to cluster these loci, we consider all substrings of the DNA sequence to be the features of expressed genes. A conventional approach is the k-mean clustering algorithm, which partitions the genomes into k clusters. Since each cluster has to capture all the adequate homologous loci, not just the parents and pseudogenes, the challenge falls in predetermining the number of clusters, k. Instead, when we examine the alignments of these substrings, we can interpret them as information flow from a source (the locus of an expressed gene) to destinations (homologous loci). These flows depend on how an aligner recognize the reads. Thus, we propose a network model with directed graphs and use a community detection algorithm to divide genomic regions into different communities based on the behavior of a specific aligner.

In a directed graph, the vertex represents a locus in the genome, and the direct edge connecting two vertices describes the information flow between two loci. The weight of each edge is depicted by the number of reads from one locus aligned to another locus. Intuitively, the heavier the weight between two vertices indicates that more features are shared between this pair of loci. An edge to itself is expected to have a high weight since most of the reads come from one gene are likely to map back to itself. On the other hand, an edge with a small weight may due to random misalignment. The information flow approach is able to remove this type of noise and exclude the vertices with weak connections in the community. Weights for this network graph can be acquired through the global distribution matrix, which keeps track of all alignment behaviors for gene features. The partitioning step uses the information from a global distribution matrix $\mathbf{X}$ to separate the genomic loci into different homologous communities. By breaking the distribution matrix into many smaller matrices, each of them can be computed independently with the same objective function described in Equation 3.4.

3.3 Read Count Profile Classification

The pseudo matrix represents ratios between the expected read count and the read count at different loci. This ratio can fluctuate slightly among replicates due to different sources of noise. To capture the noises and augment the prediction accuracy, we match the observed read count of a locus to the best profile obtained from the training data. This is performed using the k-nearest neighbors algorithm.

3.4 Training Stage

To identify the homologous loci, we use all substrings of transcript sequences as features. Given a list of transcript sequences, substrings are generated using a sliding window approach with a window size of 100bp. The choice of this window size is consistent with the read length we intend to model. These substrings are tagged with a gene ID of their origins, and are aligned back to the reference genome using TopHat2. We iterate through all alignment records to construct the distribution matrix, indicating the origins and the destinations of all reads. Since all substrings are included, this distribution matrix is referred to as the global distribution matrix, providing complete knowledge of information flows among genomic regions. We use *infomap* implemented in the *igraph* package of R [78, 79] to identify and partition homologous loci. *infomap* uses map equation as the objective function, which aims at maintaining the information flow during community partition.

For training, ten different levels of read coverage are simulated to imitate low (5X, 7X, and 10X), medium (13X, 15X, 17X, and 20X) and deep (23X, 27X, and 30X) sequencing; transcript abundance is assigned either with a fix number across all transcripts (4A, 6A, 8A) [2] or with three different sets of random numbers (R1A, R2A, R3A) [3]. In total, the combination yields 60 sets of data. Six of them are randomly chosen for validation, and the remaining sets serve as technical replicates during training. We use TopHat2 for short reads alignment. Applying its default settings, multiple alignments are reported up to 20 records, and kept in our analysis. In addition, we use Samtools [46] to retrieve the alignment information for mapped reads, and Bedtools [73] to facilitate the matching between genomic loci and gene annotations. It is worth mentioning that we target the alignment correction on the gene level, and treat isoforms as the same gene in the matching step.

3.5 Validation Stage

As described in Equation 3.4, given a new set of reads, read count for each gene can be estimated through the observed counts of all loci along with the pseudo matrices from training. The observed read counts are first matched to the best pseudo matrix using k-nearest neighbor classification, and the predicted read counts are optimized by solving the non-negative least squares equation. Both functions come from the default implementation in Matlab.

For validation, a separate set of replicates from the simulation is used to verify the predicted read counts for a list of genes. The prediction error is evaluated by the absolute relative error with respect to the true count. The true count is obtained by counting the number of reads tagged with the corresponding gene ID.

[2] We use 'A' to denote the magnitude of abundance level.

[3] Abundance levels are randomly assigned to each gene, ranging from 5A to 10A.

3.6 Read Re-assignment

After estimating read count for each gene, reads can be realigned and reassigned to the most plausible regions. The algorithm contains three phases: retain uniquely mapped reads; assign multireads to the most likely region; relocate uniquely mapped reads among homologs.

In the first phase, uniquely mapped reads that fall within the genomic locus l_j are sorted based on the sequence quality (MAPQ) and whether the mate read is properly mapped for paired-end sequencing. Each gene g_i has a corresponding locus l_i in the distribution matrix; loci that do not overlap with any expressed genes are estimated with a read count of zero. The top $\hat{y}_i$ uniquely mapped reads are kept for the associated locus l_i . If the count of retained reads reaches the estimated amount ($\hat{y}_i$), the locus is marked as resolved. On the other hand, if the number of uniquely mapped reads exceeds the estimated abundance, leftover reads are subjected to be reassigned to other homologous loci in the third phase.

In the second phase, "unresolved loci" from phase I are sorted based on their remaining counts. Unassigned multireads are retrieved for these loci and sorted using the same criteria mentioned above. The correction starts with a locus with the least amount of remaining counts, and assign the sorted multireads to locus l_j until either it is resolved or there are no more multireads aligned to this locus. Once a multiread has been assigned, it is removed from the pool. The remaining count is updated after each assignment.

The remaining unresolved loci do not have enough aligned reads, and thus require realigning leftover reads from the homologous regions identified in phase I. Within each homologous community, homologous locus can be revealed by the non-zero value in the sub-matrix of the global distribution matrix, where $x_{ij} > 0; i \neq j$ for gene g_i and locus l_j. Similar to phase II, the correction starts with the locus with the least number of remaining counts. Leftover reads are aligned back to the sequence spanned unresolved loci using Blastn [2]. The top alignments with e-value $\leq 1.00E-05$ are assigned to the unresolved gene until either there is no more alignment or the gene is resolved. The Blastn alignment record is converted to the BAM format, allowing it to proceed to downstream analyses.

CHAPTER 4

Quantification of RNA-Seq via Variable-Length *k*-mers

The lightweight algorithms for RNA-Seq quantification start with identifying a set of k-mers in a transcriptome, followed by either counting the occurrence of k-mers in RNA-Seq reads, or directly counting the reads that contain these k-mers. Since we know the transcript origins of these k-mers, transcript abundance can be inferred from read counts or k-mer counts. In our approach, we first partition a transcriptome into a set of non-overlapping clusters Θ based on sequence similarity, and select a special type of k-mers named *sig-mers*, for each cluster. These *sig-mers* represent the discriminating short sequences of a cluster and do not appear in any other clusters. Adopting the terminology introduced by RNA-Skim [110], a set of *sig-mers* S in cluster θ_i is defined as

$$\Omega(\theta_i) = \{\mathrm{S} \mid \mathrm{S} \in (k\text{-mers in } \theta_i), \forall \theta_j \in \Theta \setminus \theta_i, \mathrm{S} \notin (k\text{-mers in } \theta_j)\} \tag{4.1}$$

The size of k-mers can have a predominant effect toward the accuracy of transcript abundance estimation. We propose Fleximer, which discover and use a set of k-mers with variable length for transcript abundance estimation. We divide our objective into four components as demonstrated in Figure 4.1, with the first two focusing on *sig-mer* identification (highlighted by the blue boxes) and the last two addressing transcript quantification (highlighted by the green boxes). We follow the partition scheme in RNA-Skim to generate a set of non-overlapping clusters based on transcript sequence similarity. The partition is stored in a specialized fasta format, where each entry represents one cluster. The header starts with a cluster name, follow by transcript names. The sequence field contains all transcript sequences in a cluster, separated by a special character "|". Given this set of clusters, the transcript sequences are inspected using a suffix tree data structure to identify the *sig-mers* of each cluster. We select a subset of these *sig-mers* that are robust to read errors, and can best describe the signatures of each transcript. We then use the Aho-Corasick [1] to efficiently determine the occurrence of *sig-mers* in each read. The presence of these *sig-mers* provides the information of transcripts where each read is potentially originated from. The counts are then distributed by the EM algorithm [66] for expression quantification.

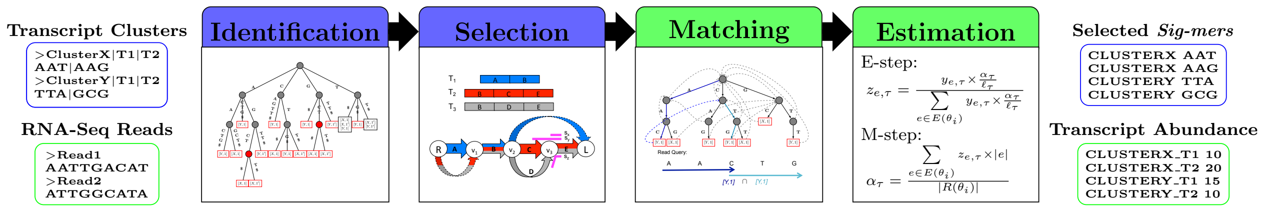


Figure 4.1: An overview of Fleximer.

4.1 *Sig-mers* Identification

Given a set of n transcripts in a cluster θ_i, our first task is to identify the *sig-mers* that can characterize the uniqueness of a cluster. Suppose the average length of transcripts is m, then the number of possible *sig-mers* with length less than k is $O(nm - nk(k+1)/2)$. In the human genome, there are over 20,000 protein coding genes, corresponding to more than 198,000 transcripts, and the average length of a transcript is approximately 2,000bp. Therefore, enumerating all possible substrings to check for their uniqueness is computationally intractable. The suffix tree is a powerful data structure for string searching algorithm, and it has been widely applied to a diverse range of biological sequence analyses [31, 36, 42]. Appendix A provides the background of suffix tree. Leveraging the suffix tree structure and properties, we apply a post-order traversal to identify all unique substrings.

We build a suffix tree based on all prefixes and suffixes of transcript sequences. We use both forward and reverse complementary sequences to construct the tree. There are two practical aspects to include the reverse complementary sequences. First, reads from the sequencing technology can be generated from the complementary strand. Second, suffixes of a reverse complementary sequence serve as the same mean as the prefixes of its forward sequence. If a substring is a *sig-mer*, then its reverse complementary sequence is also a *sig-mer*. In a suffix tree, if a substring from the root to any node appears only once in our transcript sequences, this path contains at least one *sig-mer*. We refer this node as a candidate. An internal node typically represents a substring that appears more than once in our transcript sequences. The path to any internal node contains a *sig-mer* only if all of the starting positions of this substring are located in the same cluster. These starting positions can be retrieved recursively from its descendants. Specifically, we use a bottom-up approach to examine each node through a post-order traversal.

The post-order traversal starts with the left subtree, followed by the right subtree, and the parent. We store the substring information of the nodes that have already been visited. This information does not include the actual sequence, but is sufficient to retrieve the cluster and sequence IDs, and the starting positions of a substring. At each internal node, we determine if it is a candidate by examining all substrings from its children. If all substrings come from the same cluster, then it is a candidate. Otherwise, the search process terminates for this branch, marking this node and its ancestors disqualified. Since the cluster information of a substring propagates from the leaves, all the ancestors of a disqualified node can be pruned. We use this anti-monotonic constraint to avoid further computation of its ancestors.

Figure 4.2 uses a toy example to illustrate this idea to discover all *sig-mers*. We include both forward and reverse complementary sequences of each transcript to construct a generalized suffix tree. Each leaf contains the information of cluster and sequence IDs of a suffix. Each edge is labeled by a substring. The same information of an internal node is retrieved from its descendants through a post-order traversal. Candidates are highlighted in red. For example, substring `AGCT$` and `AGCTT$` appear uniquely in sequence 1 and sequence 1 (reverse complementary of sequence 1) of cluster X, respectively. Thus, their nodes are candidates. Their parent is also a candidate since `AGCT` only appears in cluster X but not in cluster Y. However, their grandparent is not a candidate as `AG` appears in both cluster X and Y. The traversal process terminates here for this subtree. In order to generate all *sig-mers* with different sizes, we consider all prefixes of an edge label between a candidate and its parent. Therefore, `AGC` is also a *sig-mer* in addition to `AGCT`.

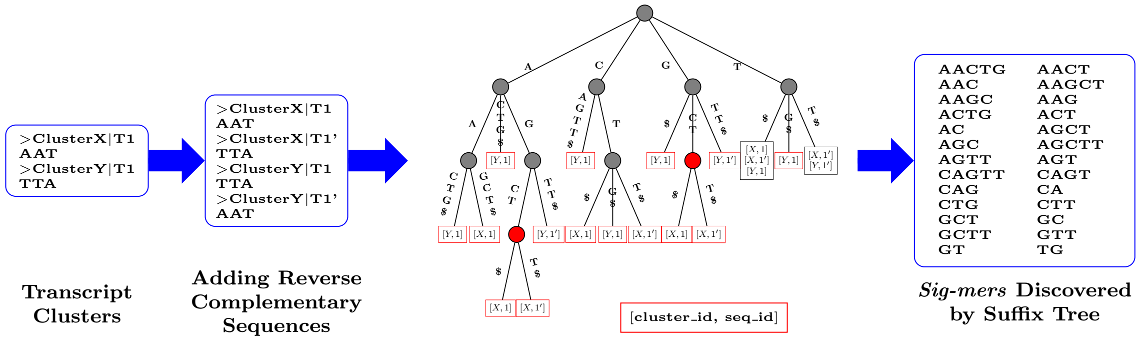


Figure 4.2: *Sig-mers* identification in Fleximer.

4.2 *Sig-mers* Selection

The compressed suffix tree allows us to efficiently discover all *sig-mers* with different sizes. However, not all of them are necessary for the quantification stage. We select a set of representative *sig-mers* that possess three properties: 1) Be robust to sequencing errors and individual variations. 2) Characterize the uniqueness of each transcript. 3) Provide sufficient coverage across all transcript sequences. We first discuss the optimal range of ks.

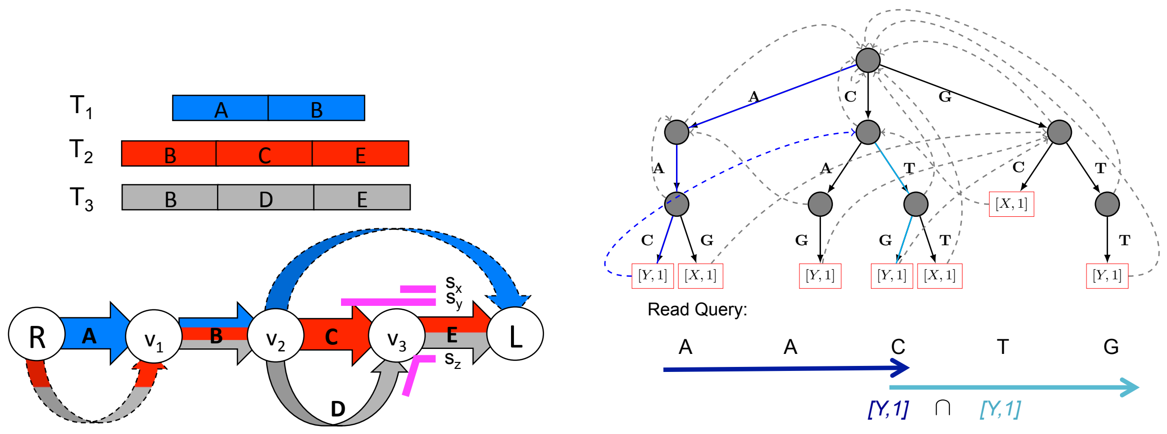
In an ideal scenario where there are no sequencing errors and individual variations, *sig-mers* will match all reads that come from the same cluster. In reality, reads contain sequencing errors and individual variations. These reads are less likely to be recognized by long *sig-mers* since the matching process requires an exact match. On the other hand, these reads have a higher risk to be "matched" by short *sig-mers* that belong to other clusters. Therefore, a *sig-mer* cannot be too long or too short, with the optimal range depending on the sequencing error rate and the fraction of individual variations. In addition, the upper bound of k is constrained by the read length of an RNA-Seq experiment. We set the lower bound to 25bp, and the upper bound to 80% of the read length.

With the remaining *Sig-mers*, we use the concept of a splicing graph [30, 83] to guide the selection. Splicing graph is first introduced to predict and represent the choices of alternative splicing for a gene model. It is a directed cyclic graph, in which vertices represent the splicing sites and edges are the exons or introns between two splicing sites. Two virtual vertices are added, root (R) and leaf (L), along with virtual edges, so that each transcript can be represented by a path that goes from R to L. Vertices with their indegree and outdegree equal to 1 are uninformative for delimiting alternative splicing event, and are often collapsed to obtain a more compact representation. Analogously, we can use a compact splicing graph to represent all member transcripts in a cluster. A cluster can contain transcripts from more than one gene, depending on the partitioning scheme. To accommodate this scenario, we broaden the definition of the edges and vertices. Edges represent different sequence segments that appear at least once in a cluster; vertices represent transition points where the incoming edge covers a different set of transcripts from the outgoing edge.

Figure 4.3a illustrates a compact splicing graph of a cluster with three transcripts. Each box represents either a unique or shared sequence segment. For example, segment B is shared by all transcripts, where segmant A is unique to transcript T_1. If they are isoforms of the same gene, segment B can be interpreted as a common exon. In this graph, each edge represents a segment, and each vertex indicates a transition point. Sequences of these three transcripts are depicted by three paths (blue, red, and gray) from R to L. A *sig-mers* is a substring of an edge, denoted by the pink lines. As we observe from the graph, a *sig-mer* possesses higher discriminating power if it spans through a vertex. Considering three *sig-mers*, s_x, s_y, and s_z, s_x resides completely on segment E and is shared by two transcripts. Given a read that contains s_x, the origin of this read can be either one of these transcripts. On the other hand, s_y and s_z are unique to transcript T_2 and T_3, respectively. Since s_y and s_z are superstrings of s_x, reads containing s_y and s_z also contain s_x. However, s_y and s_z provide specific information regarding the origin of these reads. As a result, the selection process prioritizes *sig-mers* that span through a vertex.

The original proposal of splicing graph is constructed through ESTs (expressed sequence tags) or RNA-Seq reads assembly. Since we already know the starting positions of all *sig-*

mers, we do not need to assemble these *sig-mers*. Instead, we order the *sig-mers* based on their starting positions. As we traverse through all starting positions in a sequential order, we pave all paths of a splicing graph. To provide a sufficient coverage for each transcript, we select additional *sig-mers* along the path, spaced by a small gap (e.g., 5bp) between two *sig-mers*. Since the transcript sequences can be read in either forward or reverse direction, complementary sequences of the selected *sig-mers* are added to the final list.



(a) *Sig-mer* selection via splicing graph (b) *Sig-mer* matching by TahcoRoll

Figure 4.3: *Sig-mers* selection and matching in Fleximer

4.3 RNA-Seq Reads Matching

The quantification stage starts with determining the potential transcript origins of each read. We refer this set of potential transcripts as the "transcript profile" of a read. In the traditional framework, the transcript profile is determined by aligning the read sequence to the reference transcriptome. In this setting, each *sig-mer* is associated with a transcript profile indicating its origins. We can construct the transcript profile for each read by taking the intersection of all transcript profiles associated with the *sig-mers* that appear in a read.

Discovering the occurrence of our representative *sig-mers* in reads is equivalent to the keyword searching problem in computer science. A trivial approach is to index the representative *sig-mers* (keywords) with a hash table. Since we allow variable *sig-mer* sizes, the challenge lies on scanning through each read with different window sizes. Even with an efficient hashing function, such as rolling hash [37], the number of comparisons increases with the range of window sizes. Assume that the read length is ℓ, and total length of *sig-mers* is m. The sizes of these *sig-mers* range from k_1 to k_d. The time complexity for constructing the hash is $O(m)$, and for searching *sig-mers* in one read is $O(\ell \times k_1) + \ldots + O(\ell \times k_d)$. The total time complexity would be $O(m + \ell \times \frac{k_1+k_d}{2} \times d)$.

A linear search solution is the Aho-Corasick algorithm [1], which constructs a finite state automaton for all *sig-mers*. This automaton is a keyword tree with additional links between internal nodes. These extra links allow fast transition between *sig-mer* matches without the need for backtracking. The complexity of building the automaton is $O(m)$ and for searching is $O(z)$ where z refers to the total number of occurrences of *sig-mers* in a read. Figure 4.3b illustrates the search using the Aho-Corasick algorithm. Each node consists a failure link to guide the search. Given a query `AACTG`, we follow the path `AAC` (dark blue) and find the substring belongs to sequence 1 in cluster Y. Following the failure link, we can quickly find `CTG` (light blue), which also belongs to sequence 1 in cluster Y.

A drawback of maintaining this automaton is the memory requirement for storing long or large number of *sig-mers*. As we increase the number and the length of *sig-mers*, the tree grows wider and deeper respectively. Fortunately, the concise representation of DNA molecules allows further reduction in memory requirement of this automaton. Since these *sig-mers* are composed of only four different characters: `A`, `C`, `G`, and `T`. We propose to partition these characters into two groups, and use one bit, i.e., 0 or 1, to represent them. This binarized representation allows us to significantly shrink the structure of the trie, and to substantially reduce the memory. To avoid collisions of *sig-mers* with identical binarized representations on the tree, each node contains a hash table to facilitate recovering the original *sig-mers*. We name this enhanced matching algorithm TahcoRoll, $\underline{\text{t}}$hinned $\underline{\text{Ah}}$o-$\underline{\text{Co}}$rasick automaton accelerated by $\underline{\text{roll}}$ing hash, and describe the algorithm in details below.

4.3.1 Aho-Corasick Automaton

Given a set of *sig-mers* S, and a set of reads R, our goal here is to retrieve all *sig-mers* $s \in$ S that occurs in each read $r \in$ R. This task can be reduced into multiple pattern matching [65] by mapping *sig-mers* onto patterns and reads into the input text.

Aho-Corasick algorithm (AC) conducts the matching process along a trie that corresponds to patterns. Each node in AC has a failure link that allows fast transitions from one node to the other representing its longest possible suffix without backtracking. Informally, AC constructs a finite state machine (or an automaton) that resembles a trie and failure links. The pattern matching process can be treated as transitions between nodes in the automaton, and failure links provide efficient transitions between failed matches. Figure 4.4a shows an example of AC with five *sig-mers*. Black solid links are trie links, and red dashed links are failure links. Colored nodes and thicker links are traversed while profiling a read `ATTTC`. For example, the node of *sig-mers* `ACAT` has a failure link to the node of `AT`. When profiling the read `ATTTC`, the algorithm will first match the *sig-mer* `ATT` in the *blue* node. Then, it fails to match the third `T` and transits to the *orange* node that still has no child of `T`. After traveling along the failure link again to the *yellow* node, both the last two characters `TC` can proceed towards the *orange* and *brown* nodes that indicate a match of *sig-mer* `TTC`.

The construction of the automaton in AC with *sig-mers* $s \in$ S requires a simple breadth-first search (BFS) with a $O(\sum_{s \in \mathrm{S}} |s|)$ linear time complexity. To profile *sig-mers* in read $r \in$ R, AC only needs to simulate transitions on the automaton, which also has a linear time complexity $O(\sum_{r \in R} |r| + \sum_{s \in \mathrm{S}} c_s)$, where c_s is the occurrences of s in R. The space complexity of AC is also linear, $O(\sum_{s \in \mathrm{S}|s|})$, to maintain a node and a constant number of links for each character. In theory, AC is a perfect fit for *sig-mer* profiling.

4.3.2 Thinned Automaton with Binarized Pattern Matching

Even though the theoretical bound of AC for *sig-mer* profiling is linear, there are still some hurdles in practice. One of the most critical issues is the memory usage when the number of *sig-mers* is enormous. More specifically, each individual character in *sig-mer* can be referred to as a trie node, which provides plenty information and consumes a considerable amount of memory. For example, as demonstrated in Figure 7.12b, the Python implementation of AC requires more than 240 GB of memory to process 24 million *sig-mers* whose lengths range from 131 to 151. Especially for *sig-mers* with fewer and shorter common prefixes, nodes tend to have more child nodes. The greater width leads to the increase of memory usage.

To reduce both the number of nodes and the width of the automaton, we propose the thinned automaton with binarized pattern matching. Formally, each *sig-mers* $s\left[1\ldots|s|\right] \in \mathrm{S}$ is transformed into a binarized pattern $s'\left[1\ldots|s|\right]$ before being added into the automaton. The i-th character $s'\left[i\right]$ of s' is defined as follows:

$$s'\left[i\right] = \mathrm{binarize}\left(s\left[i\right]\right), \text{ where } \mathrm{binarize}\left(c\right) = \begin{cases} 0 & , \quad c \in \{\mathtt{A}, \mathtt{G}\} \\ 1 & , \quad c \in \{\mathtt{C}, \mathtt{T}\} \end{cases}. \tag{4.2}$$

Note that these four characters can be randomly divided into two groups. From the analysis presented in Table 7.7, we use a balanced partition which groups `A,G` together. Compressing two characters into one bit 0 or 1, binarized patterns improve the representation capability of a depth-d node in a trie from 1 to 2^d unbinarized pattern(s), thereby reducing both the width of the automaton and the number of nodes. We further conduct a theoretical analysis of the improvement of this thinned automaton against the plain AC. For convenience, we assume that each character in a *sig-mer* is uniformly distributed. To estimate the worst-case scenario, we assume that every *sig-mer* has the largest length m observed in the set. While inserting a *sig-mer* into a trie, the number of newly added nodes depends on the presence of its prefixes in the trie. Proposition 1 gives an expectation of finding prefixes for n *sig-mers* with c possible characters.

Proposition 1 (Proved in Appendix B.1)**.** *Given n sig-mer with c possible characters to be added into a trie, the expected number of sig-mer that fail to find their length-i prefixes along the trie during its insertion is* $c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right) - c^{i-1}\left(1-\left(\frac{c^{i-1}-1}{c^{i-1}}\right)^n\right)$, *where* $0 \le i \le m$.

Based on Proposition 1, we derive the expected number of nodes in a trie in Proposition 2.

Proposition 2 (Proved in Appendix B.2)**.** *Given n sig-mer of length m with c possible characters to be added into a trie, the expected number of trie nodes is* $\sum_{i=1}^{m} \left[c^i - c^i \left(\frac{c^i - 1}{c^i} \right)^n \right]$.

Following Proposition 2, Proposition 3 derives the expected improvement on the number of trie nodes when the number of *sig-mers* is approaching to a large number.

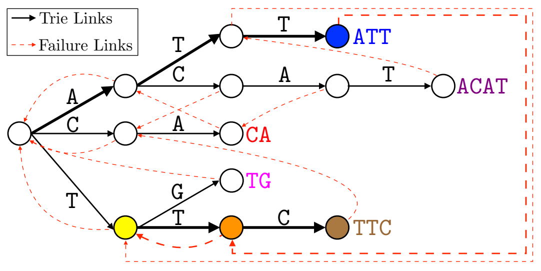
Proposition 3 (Proved in Appendix B.3)**.** *When the number of sig-mers in the automaton is approaching to a large number, the expected number of nodes in the thinned automaton is only* $\frac{3}{2} \cdot \frac{1}{2^m + 1}$ *of those in the plain AC.*

As shown in Proposition 3, the improvement with the thinned automaton is guaranteed under the assumption mentioned above. However, DNA sequences are biased. In this scenario, where the characters of each *sig-mer* are not uniformly distributed, the improvement can be more pronounced because more duplicated segments lead to fewer trie nodes. Figure 4.4b illustrates the thinned automaton of the same *sig-mers* demonstrated in Figure 4.4a.

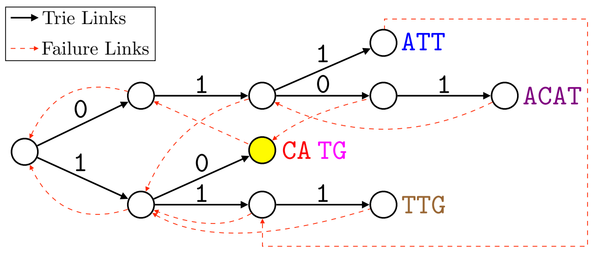
Even though the thinned automaton reduces the number of nodes, compressed representations may lead to collisions. Figure 4.5 shows an example of binarized results for five *sig-mer* and two sequencing reads. Two *sig-mers* `CA` and `TG` share the same binarized pattern `10` (highlighted in red) and result in a collision when reaching the *yellow* node in Figure 4.4b. Substrings with identical binarized representations may also lead to false matches. For instance, `ATGC` in the second read, which is not a *sig-mer*, has the same binarized representation `0101` as the *sig-mer* `ACAT` (highlighted in blue). To maintain the correctness of matching, each match to a binarized pattern needs to be verified with the original *sig-mer*. In other words, it is very time-consuming if there are serious collisions in certain nodes. A naïve comparison costs $O\left(\sum_{s \in \{\mathrm{S} \mid s' = h, s \in \mathrm{S}\}} |s|\right)$ time to verify *sig-mers* with the same representation.

4.3.3 Acceleration by Rolling Hash

Using hash functions is an intuitive idea to speed up comparisons between strings. As the lengths of *sig-mer* vary, arbitrary substrings of the read $r \in \mathrm{R}$ is required to compute



(a) The raw automaton.



(b) The binarized automaton.

Figure 4.4: The raw and binarize Aho-Corasick automaton with five signatures.

	Sig-mers S					Sequencing Reads R	
Original	ATT	CA	TTC	ACAT	TG	AATTCACAT	ATTCAGATGC
Binarized	011	10	111	0101	10	001110101	0111000101

Figure 4.5: Collisions in binarized representations.

hash values during verification. However, on-the-fly computation of hash values takes an additional linear time $O(|r|)$ for each checkup; pre-computing all possible substrings is also infeasible due to dispensable computations and extensive $O(|r|^2)$ additional memory.

To accelerate verification, we propose to apply rolling hash [13] that alleviates the time complexity for each checkup from linear to constant with a linear-time pre-processing and an additional linear memory consumption. Rolling hash is a family of hash functions where the input is hashed with a window that moves through the input. A new hash value can be rapidly calculated from the given old hash value in $O(1)$ time. It also allows $O(1)$ query time on the hash value of any substring in the input with content-based slicing. We implement the Rabin-Karp algorithm [37] as the rolling hash function. Formally, the hash value of a length-L input $r[1 \dots L]$ is defined as follows:

$$H(r[1 \dots L]) = r[1]a^{L-1} + r[2]a^{L-2} + \cdots + r[L-1]a^1 + r[L]a^0 \pmod{q}, \tag{4.3}$$

where $r[i]$ is the i-th character of the input; a is a constant multiplier; q is a constant prime modulus. The hash value of a length-i prefix of r can be recursively calculated through the hash value of the length-$(i-1)$ prefix:

$$H\left(r[1\ldots i]\right) = \begin{cases} H\left(r[1\ldots i-1]\right)\cdot a + r[i] & , \quad \text{if } i > 1 \\ r[1] & , \quad \text{if } i = 1 \end{cases} \pmod{q}. \tag{4.4}$$

With bottom-up computation, hash values of all prefixes $H\left(r[1\ldots i]\right)$ can be preprocessed in both $O(L)$ time and space complexity. Given the hash values of all prefixes, the hash value of any substring $r[i\ldots j]$ can be derived in $O(1)$ as follows:

$$H(r[i\ldots j]) = \begin{cases} H(r[1\ldots j]) - H(r[1\ldots i-1])\cdot a^{j-i+1} & , \quad \text{if } i > 1 \\ H(r[1\ldots j]) & , \quad \text{if } i = 1 \end{cases} \pmod{q}. \tag{4.5}$$

As a theoretical analysis, Proposition 4 gives a theoretical upper-bound of the collision probability. The larger the prime modulus q, the smaller the hash collision probabilities.

Proposition 4 (Gonnet and Baeza-Yates [24])**.** *The probability of two different random strings of the same length having the same hash value in Rabin-Karp rolling hash is $P(collision) \leq 1/q$, where q is the prime modulus in computations of the Rabin-Karp algorithm.*

To apply rolling hash for acceleration, each node contains a hash table that maps a hash value onto the original *sig-mers*. When transitioning to the node, the hash value of the matching substring in the read can be rapidly calculated and verified for its presence in the hash table. As a result, the average time complexity of each checkup reduces to $O(1)$. The overall time complexity of TahcoRoll is $O\left(\sum_{s\in \mathrm{S}} |s| + \sum_{r\in \mathrm{R}} |r| + \sum_{s\in \mathrm{S}} c_s\right)$, including the construction of the automaton and the matching process. The only memory overhead is hash tables with exactly $|\mathrm{S}|$ values, which is an amortized $O(|\mathrm{S}|)$ space.

4.4 Transcript Abundance Estimation

Since the transcriptome is partitioned into a set of non-overlapping cluster, each cluster can be quantified independently. After the *sig-mer* matching step, each read is associated with a transcript profile indicating its potential origins. Based on these profiles, we can group reads into their corresponding clusters. For each cluster θ_i, we use $\mathrm{R}(\theta_i)$ to represent the set of reads assigned to cluster θ_i, and $\mathrm{T}(\theta_i)$ to represent the set of transcripts in θ_i. Each transcript $\tau \in \mathrm{T}(\theta_i)$ is associated with a probability α_τ indicating its proportion of reads in cluster θ_i, and $\sum \alpha_\tau = 1$. If we know the exact origin of each read, we can form an indicator matrix $\mathbf{Z}$, where $z_{r,\tau} = 1$ indicating that read $r \in \mathrm{R}(\theta_i)$ comes from transcript τ, and 0 otherwise. Since each read only comes from one transcript, α_τ can be estimated by $\sum z_{r,\tau}$ / $|\mathrm{R}(\theta_i)|$, where $|\mathrm{R}(\theta_i)|$ is the total number of reads in θ_i. However, $\mathbf{Z}$ is not fully observed, and what we observed is the transcript profiles for each read, forming another indicator matrix $\mathbf{Y}$. Similarly, $y_{r,\tau} = 1$ if τ is in the transcript profile of read r. $\mathbf{Z}$ is the hidden variable, and is recovered from the observation $\mathbf{Y}$. We use the following likelihood function to estimate α:

$$\mathcal{L}(\alpha|\mathbf{Y}) = \prod_{r \in \mathrm{R}(\theta_i)} \sum_{\tau \in \mathrm{T}(\theta_i)} y_{r,\tau} \frac{\alpha_\tau}{\ell_\tau} = \prod_{e \in \mathrm{E}(\theta_i)} \left(\sum_{\tau \in \mathrm{T}(\theta_i)} \frac{\alpha_\tau}{\ell_\tau} \right)^{|e|} \tag{4.6}$$

In this log-likelihood function, ℓ_τ is the effective length for transcript τ. We can improve the computation speed and memory by grouping reads with the same transcript profile into equivalence classes. The concept of the equivalence class is widely used in transcript abundance estimation [9, 70, 110]. We use $\mathrm{E}(\theta_i)$ to represent the set of equivalence classes in θ_i, and equivalence class $e \in \mathrm{E}(\theta_i)$ contains $|e|$ reads. We use the EM algorithm to compute the maximum likelihood estimates of α from our observed data $\mathbf{Y}$. The algorithm alternates between allocating the fraction of counts of each equivalence class (E-step), and estimating the relative abundance given this allocation (M-step). More specifically, the E-step reconstruct the hidden variable $\mathbf{Z}$ as

$$z_{e,\tau} = \frac{y_{e,\tau} \times \frac{\alpha_\tau}{\ell_\tau}}{\sum\limits_{e \in E(\theta_i)} y_{e,\tau} \times \frac{\alpha_\tau}{\ell_\tau}}, \tag{4.7}$$

and the M-step updates the probability through $\mathbf{Z}$,

$$\alpha_\tau = \frac{\sum\limits_{e \in E(\theta_i)} z_{e,\tau} \times |e|}{|R(\theta_i)|}. \tag{4.8}$$

The algorithm converges when the change of α is less than 10^{-7}. At convergence, read count of τ can be recovered through α_τ and $|\mathrm{R}(\theta_i)|$. We report transcript abundance in three commonly used metrics: raw read count, RPKM (Reads Per Kilobase of transcript per Million mapped reads), and TPM (Transcripts Per Kilobase Million). Both RPKM and TPM are normalized units that account for sequencing depth and transcript length.

CHAPTER 5

Multifaceted Protein-Protein Interaction Prediction Based on Siamese Residual RCNN

To characterize protein-protein interaction, a model must capture the mutual influence of protein pairs. In this work, we present an end-to-end framework, PIPR, for PPI predictions using only the sequence information. The overall learning architecture is illustrated in Figure 5.1. PIPR employs a Siamese architecture of residual RCNN (recurrent convolutional neural network) encoder to better apprehend and utilize the mutual influence of two sequences. To capture the features of the protein sequences from scratch, PIPR pre-trains the embeddings of canonical amino acids to capture their contextual similarity and physicochemical properties. The latent representation of each protein sequence is obtained by feeding the corresponding amino acid embeddings into the sequence encoder. The embeddings of the two sequences are then combined using element-wise multiplication to form a sequence pair vector. Finally, this sequence pair vector is fed into a multi-layer perceptron with appropriate loss functions, suiting for specific prediction tasks. More specifically, we use the same architecture to address three challenging tasks for PPI: (i) Binary prediction to indicate whether a protein pair interacts. (ii) Interaction type prediction to identify the interaction type of two proteins. (iii) Binding affinity prediction to estimate the strength of the binding interaction.

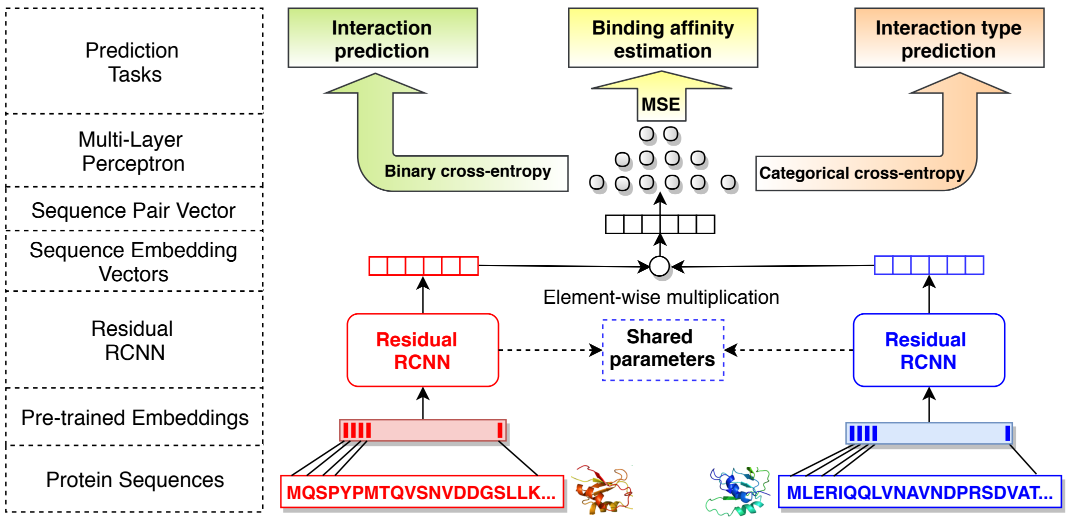


Figure 5.1: The overall learning architecture of PIPR.

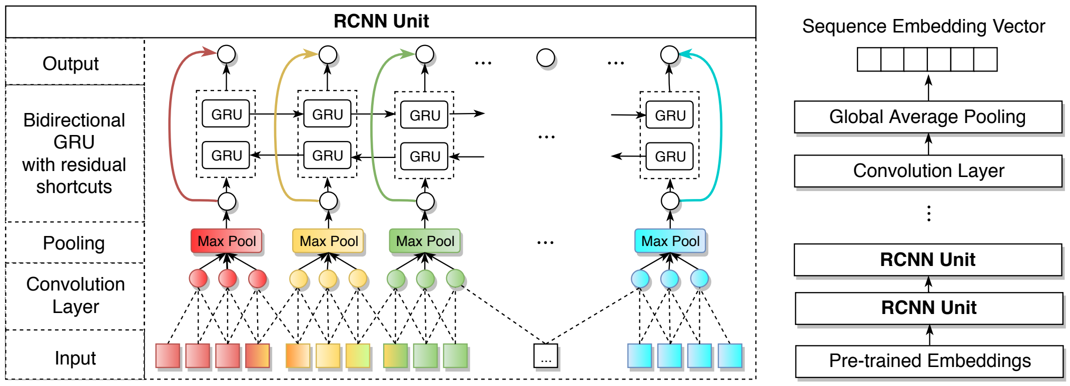


Figure 5.2: The structure of the residual RCNN encoder in PIPR.

5.1 RCNN-based Protein Sequence Encoder

We employ a deep Siamese architecture of residual RCNN to capture latent semantic features of the protein sequence pairs. In this section, we start with a brief overview of the residual RCNN architecture. The output of the stacked residual RCNN units renders the latent representation of each protein; the input of the first RCNN unit is the pre-trained amino acid embeddings. The diagram on the right of Figure 5.2 demonstrates the flow for the three components discussed in this section.

5.1.1 Residual RCNN

The RCNN seeks to leverage both the global sequential information and local features that are significant to the characterization of PPI from the protein sequences. This deep neural encoder stacks multiple instances of two computational modules, i.e. *convolution layers with pooling* and *bidirectional residual gated recurrent units*. The architecture of an RCNN unit is shown on the left of Figure 5.2.

5.1.1.1 Convolution Layer with Pooling

We use $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_l]$ to denote an input vector sequence to an RCNN unit. For the first layer of RCNN, this input vector corresponds to the embedded amino acids of the protein sequence. For the rest of the RCNN units, it corresponds to the outputs of a previous neural layer. A convolution layer applies a weight-sharing kernel $\mathbf{M}_c \in \mathbb{R}^{h \times k}$ to generate a k-dimension latent vector $\mathbf{h}_t^{(1)}$ from a window $\mathbf{v}_{t:t+h-1}$ of the input vector sequence $\mathbf{V}$:

$$\mathbf{h}_t^{(1)} = \mathrm{Conv}(\mathbf{v}_{t:t+h-1}) = \mathbf{M}_c \mathbf{v}_{t:t+h-1} + \mathbf{b}_c$$

for which h is the kernel size, and $\mathbf{b}_c$ is a bias vector. The convolution layer applies the kernel as a sliding window to produce a sequence of latent vectors $\mathbf{H}^{(1)} = [\mathbf{h}_1^{(1)}, \mathbf{h}_2^{(1)}, ..., \mathbf{h}_{l-h+1}^{(1)}]$, where each latent vector combines the local features from each h-gram of the input sequence. The n-max-pooling mechanism is applied to every consecutive n-length subsequence (i.e., non-overlapped n-strides) of the convolution outputs, which takes the maximum value along each dimension j by $\mathbf{h}_{i,j}^{(2)} = \max(\mathbf{h}_{i:n+i-1,j}^{(1)})$. The purpose is to discretize the convolution results, and preserve the most significant features within each n-stride [10, 26, 40]. By definition, this mechanism divides the size of processed features by n. The outputs from the max-pooling are fed into the bidirectional gated recurrent units in our RCNN encoder.

5.1.1.2 Residual Gated Recurrent Units

The Gated Recurrent Unit model (GRU) represents an alternative to the Long-short-term Memory network (LSTM) [12], which consecutively characterizes the sequential information without using separated memory cells [19]. Each unit consists of two types of gates to track the state of the sequence, i.e. the reset gate, $\mathbf{r}_t$, and the update gate, $\mathbf{z}_t$. Given the embedding $\mathbf{v}_t$ of an incoming item (either a pre-trained amino acid embedding, or an output of the previous layer), GRU updates the hidden state $\mathbf{h}_t^{(3)}$ of the sequence as a linear combination of the previous state $\mathbf{h}_{t-1}^{(3)}$ and the candidate state $\tilde{\mathbf{h}}_t^{(3)}$ of a new item $\mathbf{v}_t$, which is calculated as below.

$$\mathbf{h}_t^{(3)} = GRU(\mathbf{v}_t) = \mathbf{z}_t \odot \tilde{\mathbf{h}}_t^{(3)} + (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1}^{(3)} \tag{5.1}$$

$$\mathbf{z}_t = \sigma\left(\mathbf{W}_{zv}\mathbf{v}_t + \mathbf{W}_{zh}\mathbf{h}_{t-1}^{(3)} + \mathbf{b}_z\right) \tag{5.2}$$

$$\tilde{\mathbf{h}}_t^{(3)} = tanh\left(\mathbf{W}_{hv}\mathbf{v}_t + \mathbf{r}_t \odot (\mathbf{W}_{hh}\mathbf{h}_{t-1}^{(3)}) + \mathbf{b}_s\right) \tag{5.3}$$

$$\mathbf{r}_t = \sigma\left(\mathbf{W}_{rv}\mathbf{v}_t + \mathbf{W}_{rh}\mathbf{h}_{t-1}^{(3)} + \mathbf{b}_r\right). \tag{5.4}$$

$\odot$ thereof denotes the element-wise multiplication. The update gate $\mathbf{z}_t$ balances the information of the previous sequence and the new item, where capitalized $\mathbf{W}_*$ denote different weight matrices, $\mathbf{b}_*$ denote bias vectors, and σ is the sigmoid function. The candidate state $\tilde{\mathbf{h}}_t^{(3)}$ is calculated similarly to those in a traditional recurrent unit, and the reset gate $\mathbf{r}_t$ controls how much information of the past sequence contributes to $\tilde{\mathbf{h}}_t^{(3)}$. Note that GRU generally performs comparably to LSTM in sequence encoding tasks, but is less complex and requires much fewer computational resources for training.

A bidirectional GRU layer characterizes the sequential information in two directions. It contains the forward encoding process $\overrightarrow{GRU}$ that reads the input vector sequence $\mathbf{V}$ from $\mathbf{v}_1$ to $\mathbf{v}_l$, and a backward encoding process $\overleftarrow{GRU}$ that reads in the opposite direction. The encoding results of both processes are concatenated for each input item $\mathbf{v}_t$ as

$$\mathbf{h}_t^{(4)} = BiGRU(\mathbf{v}_t) = [\overrightarrow{GRU}(\mathbf{v}_t), \overleftarrow{GRU}(\mathbf{v}_t)]. \tag{5.5}$$

The residual mechanism passes on an identity mapping of the GRU inputs to its output side through a residual shortcut [29]. By adding the forwarded input values to the outputs, the corresponding neural layer is only required to capture the difference between the input and output values. This mechanism aims at improving the learning process of non-linear neural layers by increasing the sensitivity of the optimization gradients [29, 39], as well as preventing the model from the vanishing gradient problem. It has been widely deployed in deep learning architectures for various tasks of image recognition [29], document classification [14] and speech recognition [109]. In our deep RCNN, the bidirectional GRU is incorporated with the residual mechanism, and will pass on the following outputs to its subsequent neural network layer:

$$\mathbf{h}_t^{(5)} = ResGRU(\mathbf{v}_t) = [\overrightarrow{GRU}(\mathbf{v}_t) + \mathbf{v}_t, \overleftarrow{GRU}(\mathbf{v}_t) + \mathbf{v}_t] \tag{5.6}$$

In our development, the residual mechanism is able to drastically simplify the training process, and largely decreases the epochs of parameter updates for the model to converge.

5.1.2 Protein Sequence Encoding

Given a protein sequence $\mathbf{p}$, the RCNN encoder $E_{RCNN}(\mathbf{p})$ alternately stacks multiple occurrences of the above two intermediary neural network components. A convolution layer serves as the first encoding layer to extract local features from the input sequence. On top of that, a residual GRU layer takes in the preserved local features, whose outputs are passed to another convolution layer. Repeating of these two components in the network structure conducts an automatic multi-granular feature aggregation process on the protein sequence, while preserving the sequential and contextualized information on each granularity of the selected features. The last residual GRU layer is followed by another convolution layer for a final round of local feature selection to produce the last hidden states, $\mathrm{H}' = [\mathbf{h}'_1, \mathbf{h}'_2, \dots, \mathbf{h}'_{|H'|}]$. Note that the dimensionality of the last hidden states does not need to equal that of the previous hidden states. A high-level sequence embedding of the entire protein sequence is obtained from the global average-pooling [51] of H$'$, i.e. $E_{RCNN}(\mathbf{p}) = \frac{1}{|H'|}\sum_{i=1}^{|H'|}\mathbf{h}'_i$.

5.1.3 Pre-trained Amino Acid Embeddings

A protein is profiled as a sequence of amino acids, $\mathbf{p} = [a_1, a_2, \ldots, a_l]$, such that each character is an amino acid $a_i \in$ A. We use A to denote the vocabulary of 20 canonical amino acids. To support inputting the non-numerical sequence information, we use a semi-latent vector to represent each amino acid. We use bold-faced $\mathbf{a}$ to denote its embedding representation. Each embedding vector is a concatenation of two sub-embeddings, i.e. $\mathbf{a} = [\mathbf{a}_c, \mathbf{a}_{ph}]$.

The first part $\mathbf{a}_c$ measures the co-occurrence similarity of the amino acids, which is obtained by pre-training the Skip-Gram model [59] on a collection of protein sequences. The learning objective of Skip-Gram is to minimize the following negative log likelihood loss, where $|p|$ represent the sequence length of protein p.

$$J_{SG} = -\frac{1}{|p|}\sum_{a_t \in p}\sum_{-C<j<C}\log \Pr(\mathbf{a}_{c,t+j}|\mathbf{a}_{c,t}) \tag{5.7}$$

$\mathbf{a}_{c,t}$ thereof is the first-part embedding of the t-th amino acid $a_t \in p$, $\mathbf{a}_{c,t+j}$ is that of a neighboring amino acid, and C is the size of half context. The context here refers to a subsequence of a given protein sequence p, such that the length of the subsequence is $2C+1$. The probability is defined as the following softmax:

$$\Pr(\mathbf{a}_{c,t+j}|\mathbf{a}_{c,t}) = \frac{\exp(\mathbf{a}_{c,t+j}\cdot\mathbf{a}_{c,t})}{\sum_{k=1}^{n}\exp(\mathbf{a}'_{c,k}\cdot\mathbf{a}_{c,t})} \tag{5.8}$$

where n is the negative sampling size, and $\mathbf{a}'_{c,k}$ is a negative sample that does not co-occur with $\mathbf{a}_{c,t}$ in the same context.

The second part $\mathbf{a}_{ph}$ represents the similarity of electrostaticity and hydrophobicity among amino acids. The 20 amino acids can be clustered into seven classes based on their dipoles and volumes of the side chains to reflect this property. Thus, $\mathbf{a}_{ph}$ is a one-hot encoding based on the classification defined by Shen et al. [84].

5.2 Learning Architecture and Learning Objectives

5.2.1 Siamese Architecture

Given a pair of proteins $b = (p_1, p_2) \in \mathrm{B}$, the same RCNN encoder is used to obtain the sequence embeddings $E_{RCNN}(p_1)$ and $E_{RCNN}(p_2)$. Two embeddings are combined using element-wise multiplication, i.e., $E_{RCNN}(p_1) \odot E_{RCNN}(p_2)$. This is a commonly used operation to infer the relation of two sequences [26, 35, 77, 89]. Other works use the concatenation operation [87, 103], which we find to be less effective in modeling the relations of proteins.

5.2.2 Learning Objectives

The embeddings of a protein pair is fed into a multi-layer perceptron (MLP) with leaky ReLU [52] to render an output for the prediction task. This output, $\hat{y}^b$, is either a vector for a classification task or a scalar for a regression task. The learning architecture is trained to optimize the following two types of losses according to different PPI prediction problems.

(i) *Cross-entropy loss* is optimized for binary prediction and interaction type prediction. In this case, $\hat{\mathbf{y}}^b$ is a vector, whose dimensionality equals the number of classes m. $\hat{\mathbf{y}}^b$ is normalized by a softmax function, where the i-th dimension $\mathbf{y}^b = \frac{\exp(\hat{y}_i^b)}{\sum_j \exp(\hat{y}_j^b)}$ corresponds to the confidence score for the i-th class. The learning objective is to minimize the following cross-entropy loss, where $\mathbf{o}^b$ is a one-hot indicator for the class label of protein pair b.

$$L^{cross-entropy} = -\frac{1}{|\mathrm{B}|} \sum_{b \in \mathrm{B}} \sum_{i=1}^{m} o_i^b \log y_i^b \tag{5.9}$$

(ii) *Mean squared loss (MSE)* is optimized for the binding affinity estimation task. In this case, $\hat{y}^b$ is a scalar output normalized by a sigmoid function $y^b = \frac{1}{1+\exp(\hat{y}^b)}$, which is trained to approach the normalized ground truth $o^b \in [0, 1]$ by minimizing the following objective:

$$L^{MSE} = \frac{1}{|\mathrm{B}|} \sum_{b \in \mathrm{B}} \left| y^b - o^b \right|^2 \tag{5.10}$$

CHAPTER 6

Datasets

We provide a detail descriptions of the datasets used in different experiments.

6.1 NGS Datasets

6.1.1 Psuedogene Reference

The Yale pseudogene knowledgebase provides the information that describes the relationship between pseudogenes and their homologous parents. Build 79 contains 15,774 pseudogenes and 6,206 parent genes locating on the canonical chromosomes (chr1-22, X, Y, and mitochondria). Since majority of the pseudogenes are inferred computationally, and only a portion of them are presented with evidence at transcript level, we focus on these known expressed pseudogenes for the analyses. The 15,774 pseudogenes are crossed reference to the gene definitions from Ensembl release 79 of Human Genome GRCh38 [102]. Of these pseudogenes, there are only 2610 annotated as transcribed pseudogenes. Our final gene list contains 8816 transcripts (6206 parents and 2610 pseudogenes).

6.1.2 Reference Transcriptome

We use the gene annotation from Ensembl release 81 of Human Genome GRCh38 for the analyses in Fleximer and TahcoRoll, which contains more than 22,000 protein-coding genes. Of these genes, 19,817 locate on the canonical chromosomes. Our experiments focus only on the transcripts encoded by these genes, which correspond to 143,609 gene isoforms.

6.1.3 Signatures Used in TahcoRoll

To examine the effects of signature number and length, we use synthetic *k-mers* with different ranges of size, denoted by *small*(15-31bp), *medium*(65-81bp), *large*(131-151bp), and *wide*(15-131bp). Each batch contains four sets of 1.2, 6, 12, and 24 million *k-mers*. These number are arbitrarily chosen to examine the scalability of different methods. The sequence of each *k-mers* is randomly assigned with four nucleotide characters, and a random length that falls in the appropriate range. These random signatures are designed to test the worst scenario as their characters are uniformly distributed and may not share as many common prefixes as in the real sequencing data. Each *k-mer* is represented by its canonical form (i.e., the lexicographical minimum of itself and its reverse complementarysequence). Duplicated *k-mers* are removed from the list.

In order to profile real sequencing data, we use a list of 10,962,469 *sig-mers* selected by Fleximer to examine the RNA-Seq datasets, and a list of 10,935,397 short sequences randomly selected from the reference genome to examine the WGS datasets. The randomly selected signatures range from 25-60bp.

6.1.4 Simulated Reads for Pseudogene

Given a list of transcripts and their abundances, we use RNAseqSim [110] to simulate paired-end reads for training, with fragment size ranging from 100bp to 400bp. In order to mark the origin of each read, we modify the software to inherit the gene ID in read names.

6.1.5 Simulated Reads for Fleximer and TahcoRoll

We use polyester [22] to generate single-end RNA-Seq reads for selected transcripts. In each sample, 2-10% of the protein-coding transcripts are randomly selected to be expressed. Sequencing errors are implanted following the Illumina model.

In Fleximer, 15 sets of experiments with 75bp are simulated, with three different read coverages: 10X, 20X, and 30X. Each sample contains 3-97 million of reads. Each read is tagged with the name of the transcript where it originated. The expected number of reads for each transcript is computed by counting the transcript names in the raw read file. The expected RPKM and TPM are calculated from the expected read count.

In TahcoRoll, 15 sets of experiments are generated in similar manner, with five different read lengths: 75, 100, 125, 150, and 180bp. Each set contains 10-115 million of reads. The flexibility of synthetic data allows us to examne the effects of read number and length.

6.1.6 Real Data from Human BodyMap

We also include a public dataset provided by the Illumina Human BodyMap 2.0 Project (GSE30611). The samples contain individual tissue or a mixture of 16 human tissues RNA, and are sequenced using Illumina HiSeq 2000 System. Among all these samples, we use six of the single-end data with 75bp read length to evaluate the performance of Fleximer: female brain, female breast, female colon, male adrenal, male lung, and male liver. The Expression Atlas [71] provides gene expression information of highly-curated and quality-checked RNA-seq experiments, including all 16 human tissues of the Human BodyMap Project. We use the gene expression values reported in http://www.ebi.ac.uk/gxa/experiments/EMTAB- 513/ as our ground standard to evaluate the predictions of different methods.

6.1.7 Real Data from Different Sequencing Platforms

We download public datasets generated from a diverse range of sequencing platforms. The first dataset contains two experiments to study the transcriptomic analyses for lymphoblastoid cells [11]: SRR1293901 is a 2x262 cycle run from Illumina MiSeq and SRR1293901 is a 2x76 cycle run from Illumina HiSeq 2000. The second dataset, GSM1254204, aims to characterize the transcriptome of human embryonic stem cells using PacBio long reads [5]. The third datasets are generated by Oxford Nanopore to study the whole genome of breast cancer model cell line. It contains three experiments: SRR5951587, SRR5951588, and SRR5951600 with different read lengths.

6.2 Protein-Protein Interaction Datasets

6.2.1 Guo's Datasets

Guo et al. [25] generate several datasets from different species for the binary prediction of PPIs. Each dataset contains a balanced number of positive and negative samples. Among these resources, the *Yeast dataset* is a widely used benchmark by most state-of-the-art methods [26, 99, 105, 106]. There are 2,497 proteins forming 11,188 cases of PPIs, with half of them representing the positive cases, and the other half the negative cases. The positive cases are selected from the database of interacting proteins DIP_20070219 [81], where proteins with fewer than 50 amino acids or $\geq$ 40% sequence identity are excluded. We use the full protein sequences in our model, which are obtained from the UniProt [15]. The negative cases are generated by randomly pairing the proteins without evidence of interaction, and filtered by their sub-cellular locations. In other words, non-interactive pairs residing in the same location are excluded. In addition, we combine the data for *Caenorhabditis elegans*, *Escherichia coli*, and *Drosophila melanogaster* as the *multi-species dataset*. We use the cluster analysis of the CD-HIT [48] program to generate non-redundant subsets. Proteins with fewer than 50 amino acids or high sequence identify (40%, 25%, 10%, or 1%) are removed.

6.2.2 STRING Datasets.

The STRING database [88] annotates PPIs with seven types of interactions: activation, binding, catalysis, expression, inhibition, post-translational modification (ptmod), and reaction. We download all interaction pairs for *Homo sapiens* from database version 10.5, along with their full protein sequences. Among the corresponding proteins, we randomly select 3,000 proteins and 8,000 proteins that share less than 40% of sequence identity to generate two subsets. In this process, we randomly sample instances of different interaction types to ensure a balanced class distribution. Eventually, the two generated datasets, denoted by SHS27k and SHS148k, contain 26,945 cases and 148,051 cases of interactions respectively. We use these two datasets for the PPI type prediction task.

6.2.3 SKEMPI Dataset.

We obtain the protein binding affinity data from SKEMPI (the Structural database of Kinetics and Energetics of Mutant Protein Interactions) [61] for the affinity estimation task. It contains 3,047 binding affinity changes upon mutation of protein sub-units within a protein complex. The binding affinity is measured by equilibrium dissociation constant (K_d), reflecting the strength of biomolecular interactions. The smaller K_d value means the higher binding affinity. Each protein complex contains single or multiple amino acid substitutions. The sequence of the protein complex is retrieved from the Protein Data Bank (PDB) [7]. We manually replace the mutated amino acids. For duplicate entries, we take the average K_d. The final dataset contains 2,792 mutant protein complexes, along with 158 wild-types.

CHAPTER 7

Experiments and Results

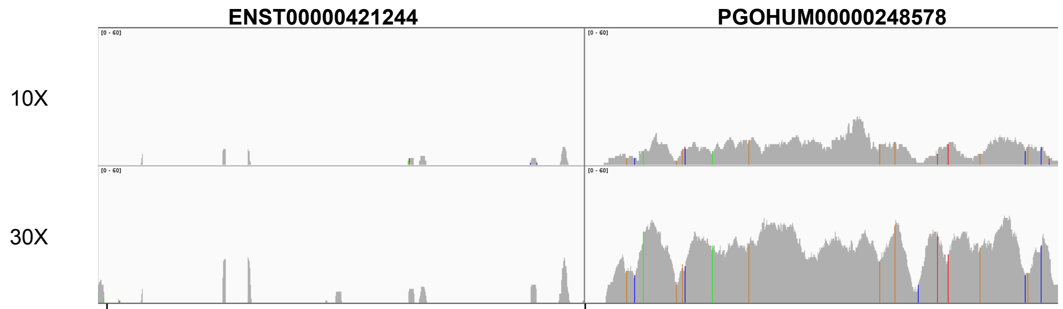
7.1 Alignment Correction for Pseudogene Abundance Estimates

Pseudogene quantification via RNA-Seq remains challenging due to the high sequence similarity with their parent genes. Short read sequences from low complexity genome region are likely to align to multiple places, and thus are difficult to re-establish the origins of the reads. In this section, we first emphasize the importance of this issue through a motivating example, followed by demonstrating the effectiveness of our approach.

7.1.1 Misalignment of Pseudogene and Its Homologous Parent

We examine the read alignments of TopHat2 [38] between homologous regions, specifically between the regions of a pseudogene and its parent. Paired-end reads are generated from one of the human transcripts for diacylglycerol kinase (DGKZ), ENST00000421244, with two different coverages, 10X and 30X. Reads are aligned to the reference genome by TopHat2, and the abundance is estimated by Cufflinks [91]. PGOHUM00000248578 is a processed pseudogenes of this transcript, and is not expressed in our simulation. Cufflinks reports a relatively high abundance in terms of FPKM (Fragment per Kilobase of transcript per Million mapped fragments) for this processed pseudogene compared to DGKZ. As illustrated in Figure 7.1, most of the reads are assigned to the pseudogene.

We further examine the relationship between expected read count and observed read count in functional parent genes and their homologous pseudogenes. We choose ten pairs of parent-pseudogenes to demonstrate the findings in Figure 7.2. Paired-end reads of 100bp



Location	11:46369137-46401497		13:44542559-44545332	
Experiment	**10X**	**30X**	**10X**	**30X**
Expected Count	109	348	0	0
Expected FPKM	2923	3105	0	0
Reported Count	9	15	153	159
Reported FPKM	160	305	3519	3673

Figure 7.1: Alignment profile between ENST00000421244 and its pseudogene.

are simulated for functional parent genes only. The expected number of reads for each gene (x-axis) is plotted against the observed read counts to itself and its pseudogene (y-axis), showing a linear relationship across different replicates. Parent gene and pseudogene profiles are depicted by different symbols, and the parent-pseudogene pair is indicated by the same color. This linear property allows us to learn a general distribution from simulated replicates.

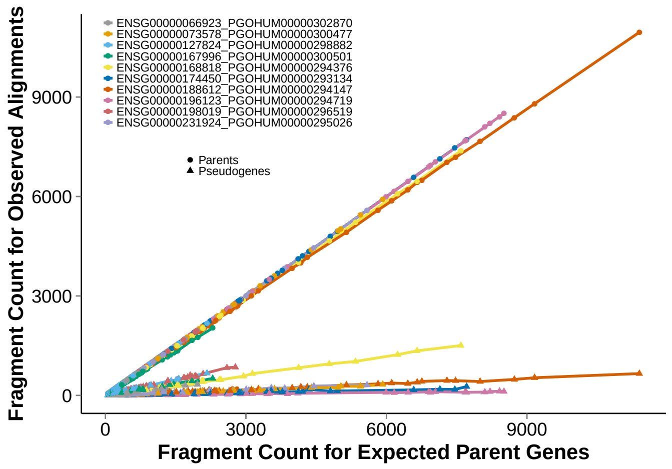


Figure 7.2: Relationship between expected and observed read counts.

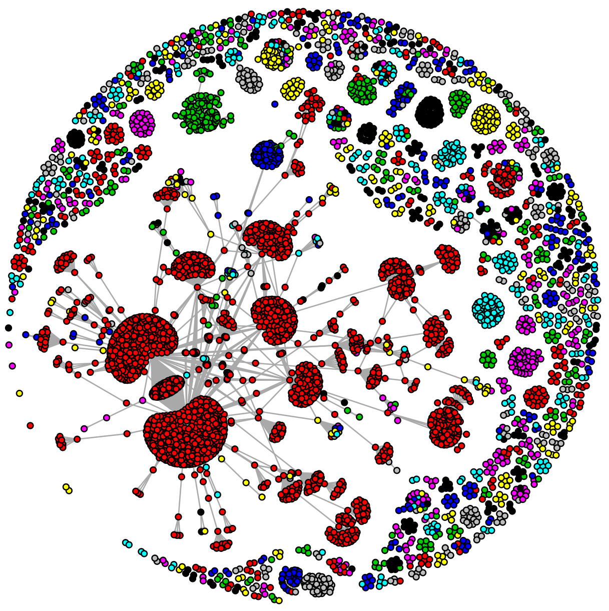
7.1.2 Community Detection for Homologous Genes

We first simulate several small subsets of parent genes and their homologous pseudogenes to examine the effectiveness and efficiency of our method. We then gradually increase the number of expressed genes to fully model the read distribution between transcribed pseudogene and their parents. Table 7.1 summarizes the datasets, including the number of expressed transcripts, corresponding genes, and the number of aligned genomic regions. Since our approach focuses on gene level, genomic regions are characterized based on gene definitions. There are 6202 parents and 2610 expressed pseudogenes in our list. For datasets A-D, parents are randomly selected from the list, along with the transcripts of their corresponding pseudogenes. Isoforms are merged and labeled with the same gene ID. Simulated reads from these genes are aligned to the reference genome. Keeping all multiple alignments, number of aligned regions are recorded for each dataset. Dataset E contains the full set of genes.

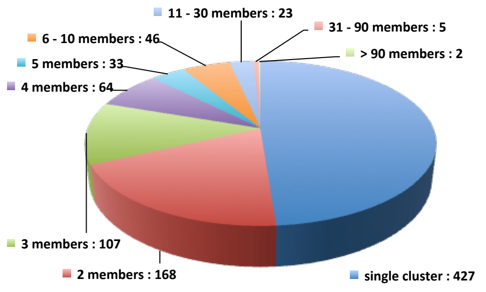
Table 7.1: Description of Simulated Datasets.

Dataset	**Parent Transcripts**	**Pseudogene Transcripts**	**Corresponding Genes**	**Aligned Loci**
A	600	304	712	4926
B	800	408	918	7146
C	1200	616	1398	8622
D	2000	930	2248	12997
E	6202	2610	5890	29071

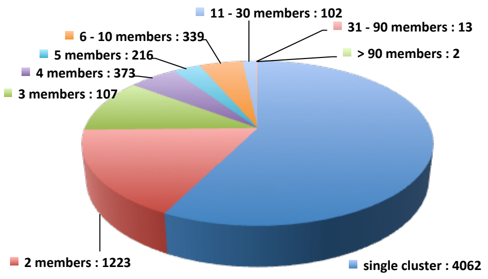
The effectiveness of the community detection technique in identifying homologous loci is first examined with a small dataset (dataset A) where 600 of parent transcripts are expressed, along with 304 pseudogene transcripts. Using the sliding window approach, consecutive substrings are treated as short reads and mapped to the reference genome. These substrings align to 4926 loci. Applying the community detection algorithm with information flow approach (*infomap*), these 4926 loci are grouped into 875 homologous communities, with the biggest community of 2377 loci. The partition of these communities is illustrated in Figure 7.3a. Loci in the same community are either positioned in a closed proximity or linked with an arrow. The arrow indicates the information flow between two loci in the same community. Figures 7.3b-c demonstrate the distribution of community size for datasets A and E respectively, where a great portion of community contains only one member. The biggest community contains 12,915 genomic loci in dataset E.



(a) Homologous community partition for dataset A.



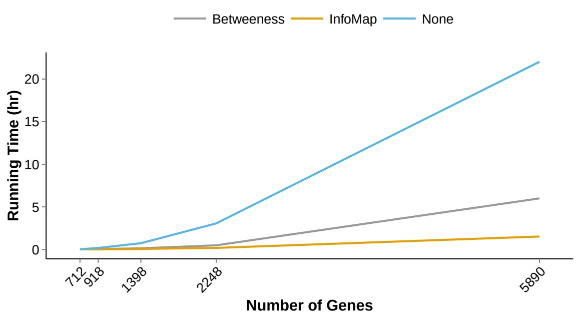
(b) Dataset A.



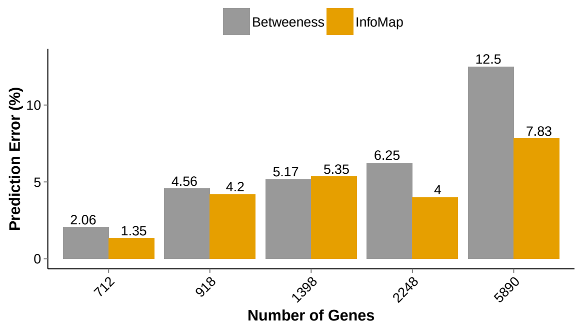
(c) Dataset E.

Figure 7.3: Homologous communities partition and size distribution.

We use all six datasets to evaluate the advantage of incorporating community partition approach to our framework. The *igraph* package in R provides two different methods to perform community detection in a directed graph, *infomap* and *edge-betweenness*. Edge-betweenness focuses on the property of community structure, and has the tendency to produce larger communities. We analyze the performance of three different approaches, including no community partition, partition with *infomap*, and partition with *edge-betweenness*. The average running time of six replicates in each dataset is plotted in Figure 7.4a. In the comparison between keeping and discarding the partition, average running time grows quadratically as we increase the number of genes without community partition. On the other hand, the running time grows linearly with the number of genes when incorporating the community detection approaches. Comparing the two community detection methods, *infomap* is able to provide a more efficient growth in running time than *edge-betweenness*. We further analyze the accuracy using the same set of data. The average prediction error is plotted in Figure 7.5b. Overall, the prediction errors are similar between *infomap* and *edge-betweenness* for smaller gene lists (dataset A - C); however, the error is much smaller (less than 10%) in *infomap* for larger gene lists (dataset D and E). Besides the advantage of time efficiency and prediction accuracy, the concept of information flow provides a better explanation to our distribution matrix, and thus is able to remove noise and capture the underlying structure of homologous gene communities.



(a) Average running time.

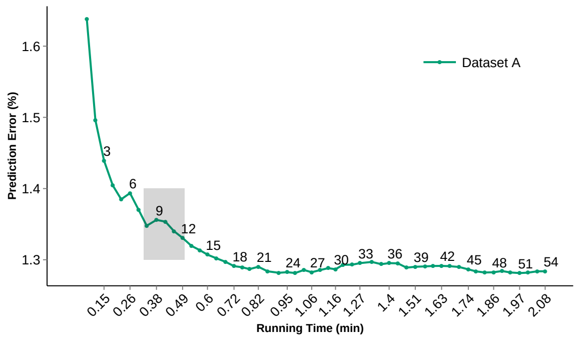


(b) Overall error.

Figure 7.4: Run time and accuracy of different community detection algorithms.

7.1.3 k-NN Classification for Homologous Community

k-NN classification is used to select the best pseudo matrix describing reads alignment behavior for a new dataset. In the training stage, there are 54 sets of replicates representing a wide range of coverage, and hence the alignment profile of each homologous community can be classified into 54 possible models. However, the running time increases as we raise the value of k, especially for datasets with a large number of expressed genes. In order to find the optimal k, we use two small gene lists (dataset A and B) to evaluate the trade-off between running time and accuracy for different values of k. The average running time and prediction errors over six replicates are plotted in Figure 7.5. The number next to each data point indicates the choice of k. The results show that the error rate drops rapidly at first, and reaches a steady phase after $k = 10$, as highlighted by the grey boxes in both figures. Compares to using all 54 replicates, a similar accuracy can be achieved at 10 replicates with a five-fold increase in speed. Consequently, we set $k = 10$ for all of the analyses.


(a) Dataset A.

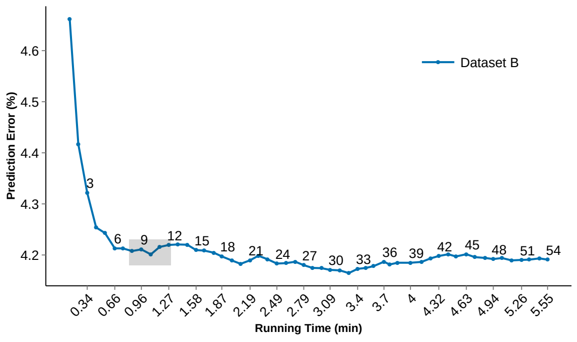

(b) Dataset B.

Figure 7.5: Parameter selection for kNN-classification.

7.1.4 Read Count Estimation for Homologous Community

As described in Chapter 3, reads are simulated for the gene list in dataset E with different coverages and abundance. Six replicates are randomly selected for validation, and the rest are used for training. For each gene community, 10 of the 54 replicates are randomly chosen for model fitting in kNN classification. Once the distribution profile is recovered, read counts are optimized through non-negative least squares estimation. We compare the read counts to the expected number of reads for these 5890 expressed genes. Table 7.2 shows the overall errors before and after applying our method. The default setting of TopHat2 keeps up to 20 multiple alignment records, and it is let to be decided by downstream analysis tools in regards to the number of multiple-alignments to keep. Read counts are quantified by counting all reads fall into gene regions, and the overall errors are above 35% for all six validations. On the other hand, our method estimates read counts through training data first, and corrects the alignments before proceeding to any downstream analysis. After applying our method on TopHat2, the error rates fall below 10% on all six validation data.

We use a small subset of genes to demonstrate our read count estimation in details in Table 7.3. The results show that the counts computed directly from TopHat2 alignments are much more deviated from the true count than applying our method. Among these genes,

Table 7.2: Overall prediction errors.

Dataset	**Validation 1 (7X, R1A)**	**Validation 2 (7X, 6A)**	**Validation 3 (7X, 8A)**	**Validation 4 (17X, R2A)**	**Validation 5 (23X, R1A)**	**Validation 6 (27X, 8A)**
TopHat2	0.374	0.357	0.353	0.382	0.363	0.363
Our Method	0.0854	0.0814	0.0806	0.0865	0.0856	0.0806

ENSG00000128422 is cross-referenced to PGOHUM00000293972, which is a homologous pseudogene of ENSG00000131885. TopHat2 reports more read alignments to this pseudogene, and less to the parent gene. Genes with more reads correctly aligned are able recovery the true read count in downstream analysis, as it is easier to remove extra alignments. However, genes with fewer alignments are less likely to be recovered since the information is missing. On the contrary, our method is able to accurately recover the true read counts among these homologous pair across all replicates.

Table 7.3: Evaluation of read count estimations.

Dataset	**Validation 1**			**Validation 2**			**Validation 3**		
Gene Name	**Truth**	**Our Est.**	**TopHat2**	**Truth**	**Our Est.**	**TopHat2**	**Truth**	**Our Est.**	**TopHat2**
ENSG00000114416	201	201.34	133	153	153.59	107	247	247.68	160
ENSG00000100522	417	415.42	310	359	340.92	255	457	450.65	341
ENSG00000134352	84	85.18	107	60	63.74	77	75	76.69	91
ENSG00000128422	780	769	1076	428	406.56	639	622	597.53	944
ENSG00000131885	193	187.89	74	158	163.46	69	218	214.36	83

Dataset	**Validation 4**			**Validation 5**			**Validation 6**		
Gene Name	**Truth**	**Our Est.**	**TopHat2**	**Truth**	**Our Est.**	**TopHat2**	**Truth**	**Our Est.**	**TopHat2**
ENSG00000114416	410	408.86	264	671	676.25	598	904	890.04	598
ENSG00000100522	889	865.56	649	1370	1368.55	1314	1734	1730.08	1314
ENSG00000134352	179	179.33	244	294	283.23	325	270	269.07	325
ENSG00000128422	857	841.52	1582	2566	2444.54	3875	2398	2487.43	3875
ENSG00000131885	402	406.97	166	677	659.43	358	837	848.37	358

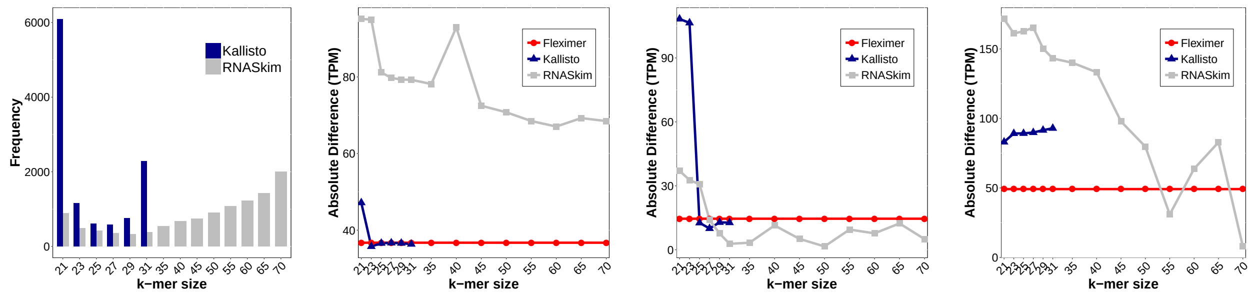
7.2 Transcript Quantification via Variable Length *k*-mers

Current lightweight quantification methods use a fixed k; however, a set of fixed size *k-mers* may not be sufficient to capture the signature of each transcript. In this section, we demonstrate the importance of having a set of *k-mers* with different sizes through a motivating example, followed by the evaluation of our method in simulated and real datasets.

7.2.1 Motivating Examples

We first evaluate the effect of different ks in transcript abundance estimation for both Kallisto [9] and RNA-Skim [110]. Kallisto allows users to set k to be any odd integer up to 31; RNA-Skim does not have any restriction on the choice of k. For Kallisto, we index the reference transcriptome using six different ks (21, 23, 25, 27, 29, 31). For RNA-Skim, we select 14 sets of *sig-mers* with different ks (21, 23, 25, 27, 29, 31, 35, 40, 45, 50, 55, 60, 65, 70). We use these *k-mers* to estimate the TPM (Transcripts Per Kilobase Million) of a simulated dataset containing 11.484 expressed isoforms. We compute the absolute differences between true and estimated TPM. A smaller value indicates a better estimation.

Among these sets of *k-mers*, we mark the k that presents the best prediction (smallest difference) for each transcript. Figure 7.6a shows the distribution of the best k. We also plot three transcripts that display interesting patterns over different ks in Figures 7.6b-d. Fleximer uses a wide range of ks, so the error (absolute difference of TPM) is plotted as a single line. The estimations fluctuate largely in RNA-Skim for these transcripts, indicating that this method is very sensitive to the choice of k. The variation is less severe in Kallisto; however, setting k to 21 produces the best prediction for ENST00000626009, but gives the worst prediction for both ENST00000265881 and ENST00000357370. These results indicate that the optimal k varies for different transcripts. The selection of k has a substantial influence on the prediction accuracy.

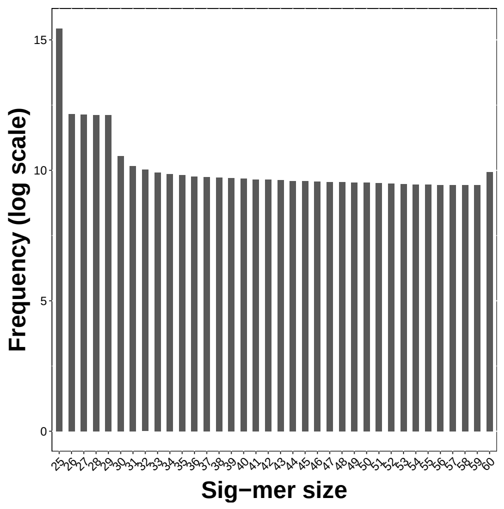


(a) *k-mer* distribution. (b) ENST00000265881. (c) ENST00000357370. (d) ENST00000626009.

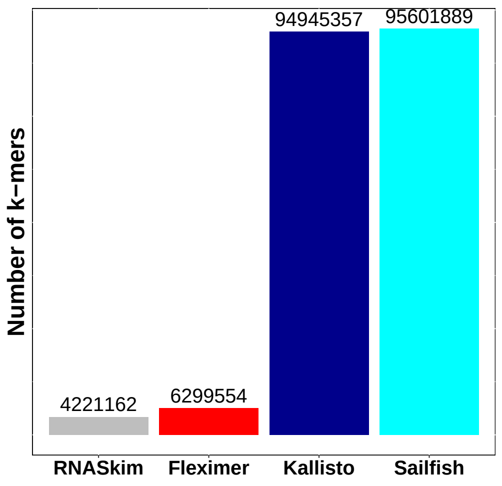
Figure 7.6: Evaluation of the TPM estimation over different ks.

7.2.2 *k-mers* Generation

Fleximer selects 6,299,554 *sig-mers* with sizes ranging from 25 to 60. Note that since the sequencing reads of our datasets are 75bp, we set the upper bound of k to 60. The distribution of the *sig-mer* size is illustrated in Figure 7.7a. The majority of the *sig-mers* reside on our lower boound; however, the rest of the *sig-mers* are equally distributed among this range. On the other hand, using the default parameters, RNA-Skim generates a set of 4,221,162 *sig-mers* with a fixed size of 60. Using the recommended k of 31, Sailfish produces a set of 95,601,88 *k-mers* and Kallisto produces 94,945,357 *k-mers*. Figure 7.7b compares the number of *k-mers* generated in different methods. In terms of computational resources, Fleximer needs more time to identify a set of optimal *sig-mers* than other methods since the search space is much bigger. It is worth mentioning that this step only needs to be executed once. Sadakane's compressed suffix tree allows us to perform this step in less than 10G, which requires less memory than Sailfish and RNA-Skim.



(a) *Sig-mer* size distribution of Fleximer.



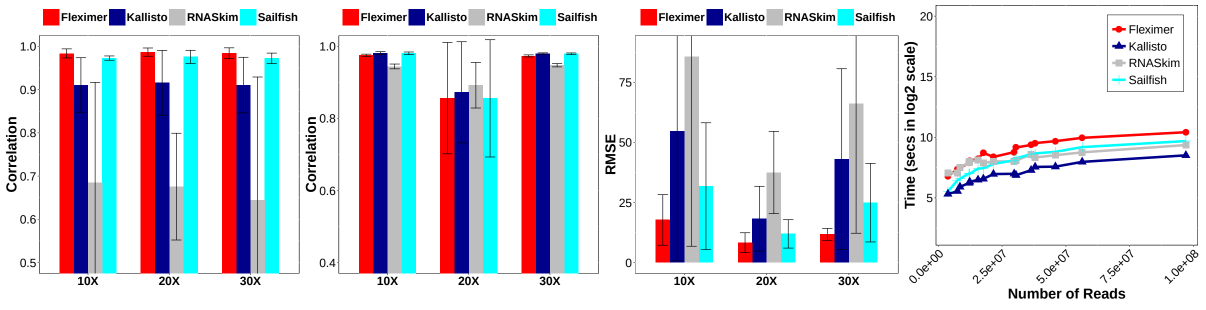
(b) Number of *k-mers*.

Figure 7.7: *K-mer* statistics of different methods.

7.2.3 Simulation Study

We evaluate the prediction accuracy using 15 simulated datasets. The accuracy metrics include Pearson correlation, rank correlation, and root-mean-square error (RMSE).

Figurse 7.8a-c illustrate three metrics separately for different read depths (10X, 20X, 30X). Each bar shows the mean and standard deviation over five datasets. Results indicate that Fleximer and Sailfish perform similarly with consistent Pearson and rank correlations. The Pearson correlations are slightly higher for Fleximer at different read depths. Higher correlation values are better as the predictions are in the same trend as the ground truth. On the other hand, lower residual errors are better as the predictions are closer to the ground truth. Fleximer presents the smallest RMSE among four methods. We also evaluate the running time as shown in Figure 7.8d. Fleximer requires more time than others due to searching *sig-mers* of different sizes; however, it scales linearly with the number of reads. All experiments finish in less than 25 minutes (1500s). We further examine the utilization of our representative *sig-mers*. We calculate the fraction of reads recovered by each method. Figure 7.9a shows that Fleximer is able to recognize more than 90% of the reads with fewer *k-mers* than Kallisto and Sailfish. The *sig-mers* used in Fleximer are much more effective than the *sig-mers* in RNA-Skim, which only recovers less than 30% of the reads.



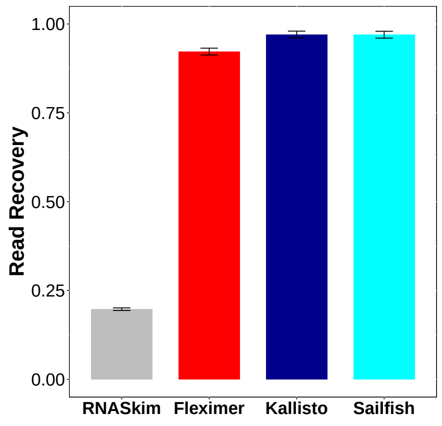
(a) Pearson Correlation. (b) Rank Correlation. (c) RMSE. (d) Run Time.

Figure 7.8: Accuracy and efficiency evaluations using simulated data.

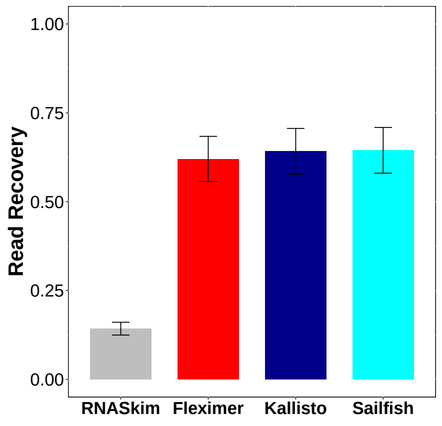
7.2.4 Human BodyMap 2.0 Project

Simulated datasets cannot capture all of the biases presented in a real RNA-Seq experiment; therefore, we evaluate the performance of different methods using six datasets from the Human BodyMap project. Figure 7.9b shows that on average, Fleximer is able to identify a similar amount of reads in these datasets as Sailfish and Kallisto ($> 60\%$), but with fewer number of *k-mers*. This result is consistent as demonstrated in the simulated studies.

Since we do not know the true abundance of each transcript, we use the estimated abundance reported in Expression Atlas as our ground standard. Expression Atlas provides expression values in RPKM or FPKM (Read or Fragment per Kilobase of transcript per Million mapped fragments) on gene-level, instead of transcript level. Two post-processing techniques are applied to normalize these values. First, we convert expression values reported in Expression Atlas to TPM. Second, we aggregate the expressions of all gene-isoforms estimated in each method to represent the gene expression. After normalization, we remove the non-protein coding genes from Expression Atlas and perform the analyses on gene-level. Figure 7.10 shows six Venn diagrams to compare the number of expressed genes identified by each method. Expressed genes are defined as those with TPM greater than zero. Across six datasets, all methods identify a large number of shared expressed genes as demonstrated



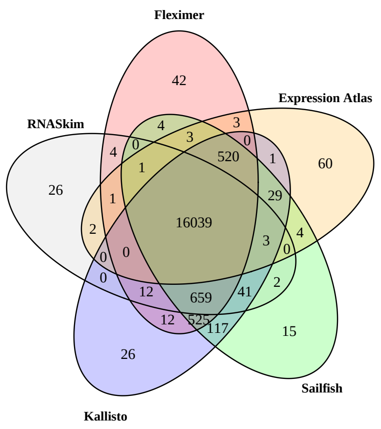
(a) Simulated Studies.



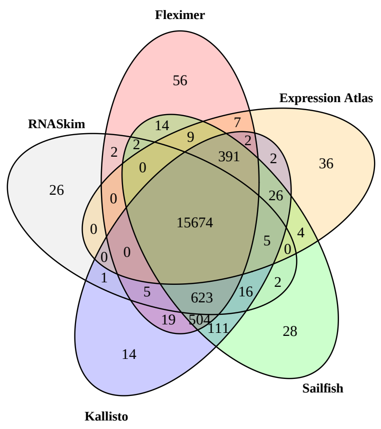
(b) Human BodyMap.

Figure 7.9: Read recovery rate of different methods.

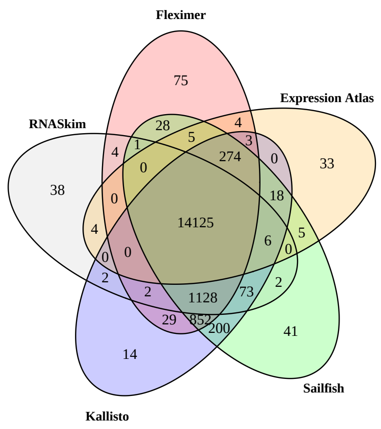
by the large overlap in Venn diagrams. Among these four methods, RNA-Skim misses out the largest number of expressed genes reported in Expression Atlas (counts that are outside the RNA-Skim ellipse and inside the Expression Atlas ellipse: 620 for adrenal, 477 for lung, 342 for liver, 532 for brain, 472 for breast, and 407 for colon). Each method identifies a few number of unique genes that are not picked up by others. However, Sailfish and Kallisto pick up around 100 more genes than Fleximer that are not reported in Expression Atlas (105 and 134 more for Sailfish and Kallisto respectively in adrenal, 75 and 68 more in lung, 206 and 181 more in liver, 121 and 126 more in brain, 104 and 99 more in breast, and 125 and 129 more in colon). These false positives are illustrated by comparing the counts outside the Expression Atlas ellipse. We also experiment with different thresholds in defining the expressed genes (i.e. 0.1, 0.5, and 1), and observe similar results as setting it to 0.



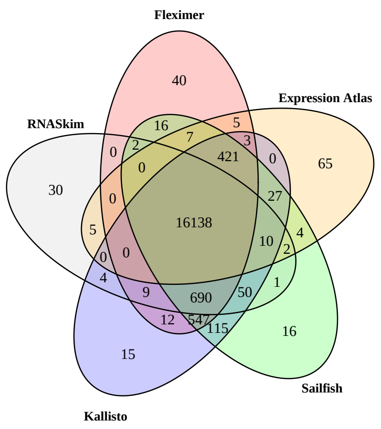
(a) Male Adrenal.



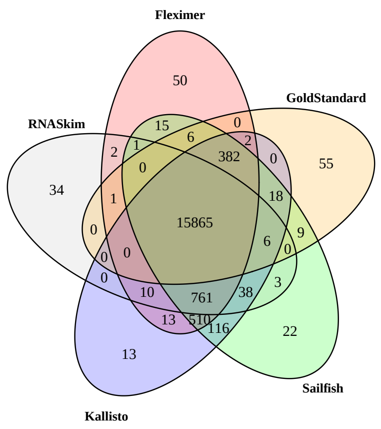
(b) Male Lung.



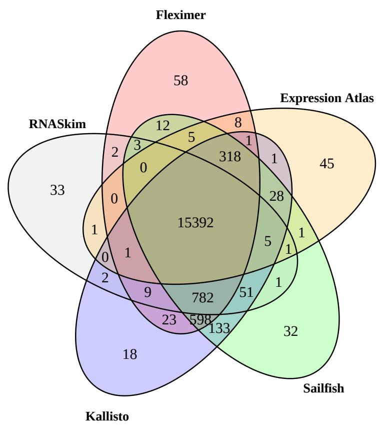
(c) Male Liver.



(d) Female Brain.



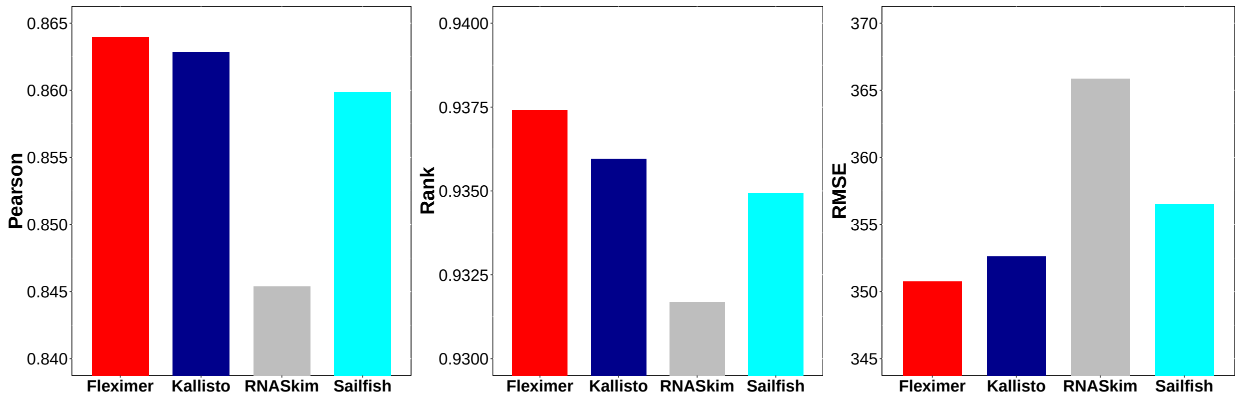
(e) Female Breast.



(f) Female Colon.

Figure 7.10: Evaluation of Human BodyMap with Venn diagrams.

We further evaluate the prediction power of four different methods against the values reported in Expression Atlas using all genes. Figure 7.11 summarizes the average values of each accuracy metric over six datasets on gene-level. Fleximer demonstrates the best accuracy assessments, with the highest Pearson and rank correlations and the lowest RMSE.



(a) Pearson Correlation. (b) Rank Correlation. (c) RMSE.

Figure 7.11: Evaluation of Human BodyMap with accuracy metrics.

7.3 Signature Profiling via Thinned Aho-Corasick Automaton

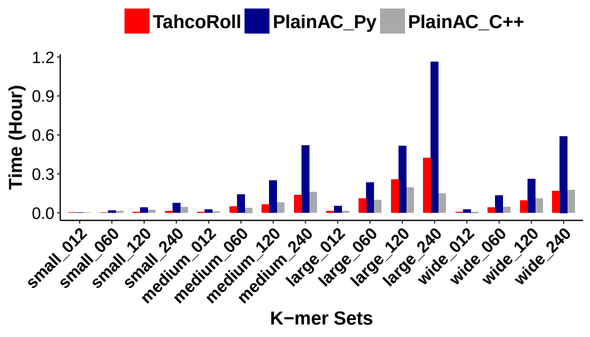
Aho-Corasick (AC) algorithm presents a promising approach to profile a list of variable-length *k-mers* in genomic data. However, the memory consumption of the automaton can be impractical as we increase the number and the size of *k-mers*. To mitigate the issue, we propose TahcoRoll which builds upon the Aho-Corasick algorithm for this task. Specifically, we partition nucleotides into two groups and use one bit to represent them. We integrate the rolling hash technique to efficiently recover the original string for matching. In this section, we demonstrate the effectiveness of our approach through both synthetic and real data. The analyses focus on counting the occurrences of a list of signatures (*k-mers*) in the dataset.

7.3.1 Experimental Settings

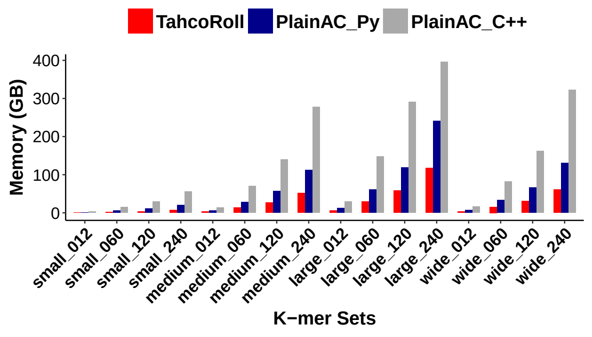
Since most of the *k-mer* counters process reads with a fixed k, we use a Python script as a wrapper to handle different *k-mer* sizes and to call the appropriate functions from command line. We include all the *k-mer* counters mentioned in the related works (Chapter 2). We implement two baseline methods. The first one is a naïve implementation in `C++`, denoted by "Naïve". It uses a hash table to store *k-mers* and scans through the reads multiple times with different window sizes. Theoretically, Naïve is light in memory, but requires an extensive running time. The second baseline is the conventional Aho-Corasick algorithm. We test two public source code written in `Python` (PlainAC_Py) and `C++` (PlainAC_C++). Configuration details of different softwares are described in Appendix D.

7.3.2 Automaton Construction

The memory of AC is sensitive to the composition of signature patterns, such as *k-mer* lengths, the number of *k-mers*, and common prefixes shared by different *k-mers*. Figure 7.12 compares the computational resources used for automaton construction in different approaches over 16 sets of signatures. The implementation of PlainAC_C++ uses several additional data structures on each node to facilitate the traversal, causing a huge memory overhead. As a result, PlainAC_C++ is fast, but requires twice and five times more memory than PlainAC_Py and TahcoRoll, respectively. For the large batch of 24 million *k-mers*, PlainAC_C++ maxes out the memory capacity (>396GB) of our server, and thus the recorded time is truncated. Our thinned automaton consistently requires less time than PlainAC_Py in construction. As we increase the number of *k-mers*, the construction time rises. The memory of the thinned automaton is significantly reducing to nearly half of the memory required in PlainAC_Py. Larger *k-mers* are more diverse in their sequences, and often share shorter common prefixes with others. This phenomenon is reflected in our analysis, where the *large* batches of *k-mers* require more time and memory than others. The *wide* batches use less resource than the *large* ones because they contain fewer long *k-mers*.



(a)



(b)

Figure 7.12: Run-time and Memory for automaton construction.

7.3.3 Pilot Study of 13 Approaches

We perform a preliminary assessment of the memory footprint and running time on 11 existing counters, together with two baselines and TahcoRoll. Since most of these counters are designed to process data with a fixed k, they present a limitation on handling a wide range of k's. Moreover, few of these algorithms limit the choice of k. For these reasons, we inspect the capability of different approaches with single thread using small datasets: synthetic reads with 75bp and small batches of signatures.

We separate the analyses into two panels as demonstrated in Figure 7.13. The top panel focuses on different number of reads with 1.2 million of k-*mers*, and the bottom panel focuses on different number of k-*mers* with 34,497,448 reads. Methods in the bottom-left corner of each plot indicate being both time and memory efficient. As we predicted, Naïve uses very little memory, but takes a long time to complete. PlainAC is fast, but requires a large amount of memory when increasing the number of k-*mers*. Consistent with the analysis in Figure 7.12, PlainAC_C++ uses twice as much memory as PlainAC_Py.

TahcoRoll is the most efficient approach in five out of these six analyses. KMC3 and Squeakr use less memory, but requires more time than TahcoRoll when there are 24 million k-*mers*. When we fix the number of k-*mers* (top panel), the memory footprint and running time for KMC3 and Squeakr increase with the number of reads, but the memory stays constant for both TahcoRoll and Jellyfish. Jellyfish is memory efficient when counting a given list of *k-mers* with the same size; however, repeating this process for different k's makes it more time-consuming than TahcoRoll.

We also validate the accuracy by comparing the counts reported in each approach to Naïve. Probabilistic approaches, BFCounter and khmer, neglect singleton *k-mers*. The exact counting mode of Squeakr is unable to count small *k-mers* if all of them exist in reads, due to its implementation design. All other approaches agree with Naïve.

To avoid waiting on extremely slow counters, we remove Naïve, DSK, khmer, BFCounter, MSPKC, and KCMBT for further analyses. From the memory usage prospectie, we take out Tallymer and PlainAC_C++ since Tallymer does not scale well with more and longer reads and PlainAC_C++ uses up the memory for more and longer *k-mers*. Squeakr is also removed as it often fails to count smaller *k-mers*. The remaining analsyses are carried out for PlainAC_Py, Jellyfish, KMC3, MSBWT, and TahcoRoll.

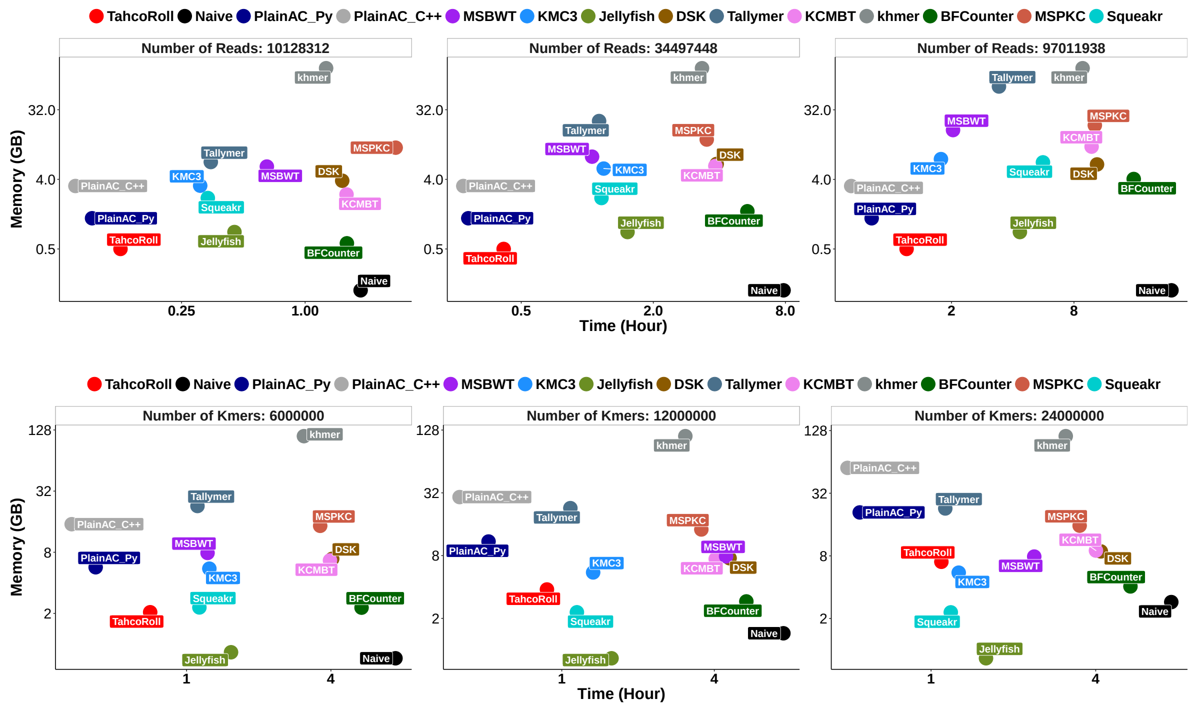


Figure 7.13: Run-time and memory for pilot study.

7.3.4 Extensive Study on Synthetic Datasets

We use 1.2 million *k-mers* ranging from 15-151bp (*wide*) to evaluate the scalability on different read lengths and number of reads. We highlight the total run time (hour) and memory consumption (GB) of each approach in Table 7.4. Time is further split into the preparation (Prep) and querying (Query) phases. In TahcoRoll and PlainAC_Py, the preparation phase refers to automaton construction and the querying stage queries all reads. In MSBWT and KMC3, the preparation stage focus on indexing the reads, and the querying stage queries all *k-mers*. Therefore, running time of query phase is not in the same scale across different approaches. We use Jellyfish's function to count the list of *k-mers* directly, so the running time cannot be split in details. Dagger (†) marks the most time efficient approach; asterisk (∗) marks the most memory efficient approach. Although Jellyfish is the second most memory efficient approach behind TahcoRoll, its running time does not scale well with datasets containing more or longer reads. TahcoRoll consistently outperforms others across different read sets in both time and memory. Our thinned automaton is more compact, which makes it more efficient than the conventional automaton.

Table 7.4: Run-time and memory of different read sets.

Read	**Total**	**TahcoRoll**				**PlainAC_Py**				**MSBWT**				**KMC3**				**Jellyfish**	
Length	**Reads**	Prep	Query	**Time**	**Mem**	Prep	Query	**Time**	**Mem**	Prep	Query	**Time**	**Mem**	Prep	Query	**Time**	**Mem**	**Time**	**Mem**
75bp	10,128,312	0.006	0.09	**0.10**†	**3.29***	0.02	0.11	**0.13**	**6.75**	0.40	0.01	**0.41**	**5.86**	1.49	0.02	**1.51**	**3.30**	**1.83**	**4.74**
	34,497,448	0.005	0.28	**0.29**†	**3.29***	0.02	0.37	**0.39**	**6.75**	0.90	0.01	**0.91**	**7.88**	2.45	0.09	**2.53**	**5.52**	**5.55**	**4.74**
	97,011,938	0.005	0.78	**0.78**†	**3.29***	0.02	1.04	**1.06**	**6.75**	3.26	0.02	**1.95**	**17.57**	5.82	0.68	**6.50**	**8.00**	**15.24**	**4.74**
100bp	11,397,007	0.005	0.13	**0.14**†	**3.29***	0.03	0.17	**0.20**	**6.75**	0.45	0.01	**0.46**	**6.40**	2.42	0.33	**2.75**	**3.83**	**3.32**	**4.74**
	41,054,662	0.005	0.47	**0.48**†	**3.29***	0.02	0.59	**0.61**	**6.75**	1.29	0.01	**1.30**	**11.10**	4.81	0.61	**5.42**	**7.56**	**11.25**	**4.74**
	114,813,452	0.006	1.35	**1.36**†	**3.29***	0.02	1.59	**1.61**	**6.75**	3.49	0.02	**3.51**	**26.00**	16.23	3.35	**19.58**	**20.10**	**31.78**	**4.74**
125bp	10,822,319	0.004	0.15	**0.15**†	**3.29***	0.03	0.19	**0.22**	**6.75**	0.63	0.01	**0.65**	**6.83**	2.81	0.81	**3.61**	**4.15**	**5.26**	**4.74**
	58,012,701	0.005	0.77	**0.78**†	**3.29***	0.03	0.99	**1.02**	**6.75**	2.59	0.02	**2.61**	**17.48**	10.53	2.51	**13.04**	**19.03**	**27.22**	**4.74**
	107,375,244	0.005	1.37	**1.38**†	**3.29***	0.02	1.84	**1.87**	**6.75**	4.56	0.02	**4.58**	**29.75**	18.46	3.92	**22.37**	**34.59**	**50.41**	**4.74**
150bp	27,628,054	0.006	0.35	**0.36**†	**3.29***	0.02	0.55	**0.57**	**6.75**	1.69	0.01	**1.71**	**11.46**	9.09	1.88	**10.97**	**14.87**	**18.78**	**4.74**
	57,437,772	0.007	1.20	**1.21**†	**3.29***	0.02	1.20	**1.22**	**6.75**	3.50	0.02	**3.51**	**20.31**	17.26	3.98	**21.24**	**31.10**	**36.86**	**4.74**
	114,306,300	0.006	2.01	**2.01**†	**3.29***	0.03	2.42	**2.44**	**6.75**	5.86	0.02	**5.88**	**37.27**	33.45	8.25	**41.69**	**58.23**	**74.29**	**4.74**
180bp	16,197,631	0.006	0.35	**0.35**†	**3.29***	0.03	0.40	**0.43**	**6.75**	2.43	0.01	**2.45**	**9.30**	7.45	1.99	**9.44**	**14.61**	**15.51**	**4.74**
	37,836,905	0.005	0.86	**0.87**†	**3.29***	0.02	0.87	**0.90**	**6.75**	3.20	0.02	**3.22**	**16.96**	16.05	4.20	**20.26**	**33.34**	**35.14**	**4.74**

Next, we use 86,976,737 reads of 180bp to evaluate the scalability on different batches of

Table 7.5: Run-time and memory for profiling different signature sets.

K-mer Batch	**Total K-mers**	**TahcoRoll**				**PlainAC_Py**				**MSBWT**				**KMC3**				**Jellyfish**	
		Prep	Query	**Time**	**Mem**	Prep	Query	**Time**	**Mem**	Prep	Query	**Time**	**Mem**	Prep	Query	**Time**	**Mem**	**Time**	**Mem**
Small (15-31bp)	1,200,000	0.0004	2.56	**2.56**	**0.51***	0.003	1.81	**1.81**†	**1.25**	5.39	0.01	**5.40**	**34.35**	3.99	0.35	**4.34**	**14.61**	**11.15**	**0.83**
	6,000,000	0.002	4.83	**4.83**	**2.09**	0.02	2.42	**2.44**†	**5.70**	5.39	0.06	**5.46**	**34.31**	5.39	0.84	**6.23**	**14.61**	**11.11**	**0.83***
	12,000,000	0.003	5.48	**5.48**	**3.85**	0.03	2.74	**2.77**†	**10.93**	5.39	0.12	**5.51**	**34.35**	5.89	0.95	**6.84**	**14.61**	**11.17**	**0.83***
	24,000,000	0.006	7.22	**7.23**	**7.13**	0.09	3.11	**3.21**†	**20.93**	5.39	0.23	**5.63**	**34.35**	5.94	0.96	**6.91**	**14.61**	**11.15**	**0.83***
Medium (65-81bp)	1,200,000	0.005	2.01	**2.01**†	**2.82**	0.03	2.42	**2.45**	**5.83**	5.39	0.01	**5.40**	**34.35**	5.03	2.59	**7.62**	**58.16**	**11.41**	**2.47***
	6,000,000	0.02	2.47	**2.49**†	**13.49**	0.13	4.77	**4.90**	**28.59**	5.39	0.06	**5.45**	**34.35**	4.90	1.91	**6.81**	**58.16**	**11.37**	**2.47***
	12,000,000	0.09	3.53	**3.62**†	**26.52**	0.27	5.27	**5.54**	**56.71**	5.39	0.11	**5.50**	**34.33**	4.90	1.93	**6.83**	**58.16**	**11.07**	**2.47***
	24,000,000	0.16	4.00	**4.16**†	**52.11**	0.75	5.25	**6.00**	**112.5**	5.39	0.22	**5.61**	**34.35**	4.87	1.49	**6.37**	**58.16**	**11.36**	**2.47***
Large (131-151bp)	1,200,000	0.02	2.65	**2.67**†	**6.10**	0.06	2.98	**3.04**	**12.24**	5.39	0.02	**5.41**	**34.35**	3.51	2.35	**5.87**	**67.27**	**6.66**	**4.74***
	6,000,000	0.08	3.51	**3.59**†	**29.91**	0.29	4.34	**4.63**	**60.63**	5.39	0.08	**5.47**	**34.35**	4.28	4.04	**8.32**	**67.27**	**6.72**	**4.74***
	12,000,000	0.18	4.37	**4.55**†	**58.43**	0.55	4.98	**5.53**	**118.97**	5.39	0.16	**5.55**	**34.33**	5.14	4.50	**9.65**	**69.19**	**8.59**	**4.38***
	24,000,000	0.42	4.42	**4.84**†	**117.79**	1.33	4.90	**6.23**	**240.67**	5.39	0.29	**5.69**	**34.35**	4.10	3.05	**7.17**	**67.27**	**6.73**	**4.74***

k-mers, which are designed to test the worst scenario. Table 7.5 shows that when the *k-mers* are short (*small* batch), PlainAC_Py uses the least amount of time. This observation is due to less collision in the signature sets. Under a severe condition where there is a large number (12 and 24 million) of *k-mers* with uniformly distributed characters, TahcoRoll requires more memory than MSBWT in three out of six cases. It is worth mentioning that both MSBWT and KMC3 write a huge amount of intermediate files to disk (at least 16GB for MSBWT and 43 GB for KMC3 in this dataset) to alleviate the memory bottleneck. In contrast, TahcoRoll is an in-memory approach that does not generate any intermediate data.

MSBWT, KMC3, and Jellyfish allow indexing reads in parallel, so we evaluate the parallel settings on the *wide* batch of *k-mers*. Figure 7.14 shows the running time of analyzing 86,976,737 synthetic reads of 180bp across four sets of *k-mers*. Both Jellyfish and TahcoRoll scale well with number of threads, but the improvement of MSBWT and KMC3 is marginal. This is mainly due to the limitation of I/O as these two approaches constantly read and write files to disk. The running time of TahcoRoll remains faster than others across different experiments and threads. The four-thread TahcoRoll also demonstrates to be faster than others with 16 threads.

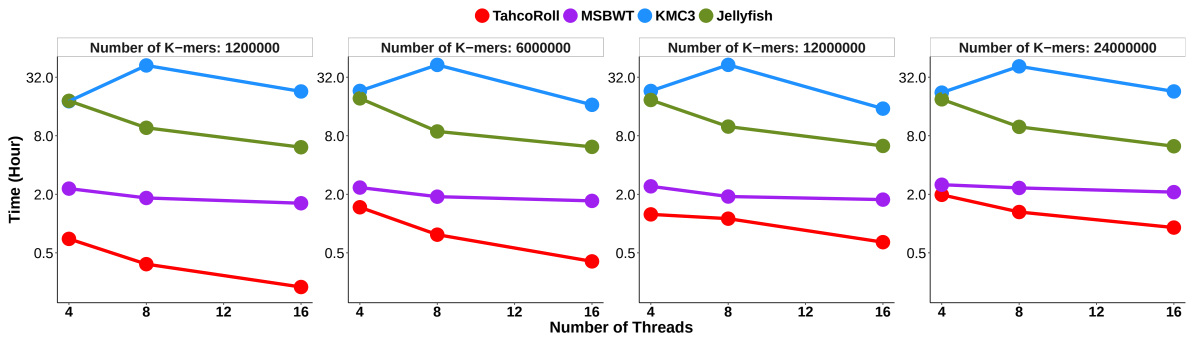


Figure 7.14: Run-time for parallel study.

7.3.5 Real Datasets from Different Sequencing Platforms

Synthetic studies demonstrate the worst case of signature sets. Here, we examine the practical usage by analyzing signatures from real DNA sequences with reads from different sequencing platforms. Experiments are conducted on a desktop machine of Core^{tm} i7-3770 CPU@3.4.0GHz.

Table 7.6 summarizes the nature and analysis of each dataset. MSBWT, KMC3, and Jellyfish are run with eight threads; TahcoRoll is run with single thread and eight threads. For the measurement that is less efficient than TahcoRoll, we compute the fold-change to those reported by the eight-thread TahcoRoll. MSBWT is unable to finish indexing for the PacBio data within two days. KMC3 cannot index long reads from Nanopore as it exceeds the buffer size automatically set by the program; it also uses up the memory on the machine (32G) for the PacBio data. Overall, the running time of single-thread TahcoRoll is as efficient as Jellyfish with eight threads, and significantly outperforms KMC3 and MSBWT in short and long reads, respectively. In the parallel settings, TahcoRoll runs at least four times faster than MSBWT and Jellyfish, and demonstrates a drastic improvement over KMC3.

Lastly, we examine the impact of different binary representations of the nucleotides. The concise representation requires a many-to-one mapping between four nucleotides and two single binary values. The four characters can be divided into balanced partitions: [{`A,C`}, {`G,T`}], [{`A,T`}, {`C,T`}], and [{`A,G`}, {`C,T`}], or unbalanced partitions: [{`A`}, {`C,G,T`}], [{`C`}, {`A,G,T`}], [{`G`}, {`A,C,T`}], and [{`T`}, {`A,C,G`}]. The default setting groups {`A,G`} together,

Table 7.6: Run-time and memory of real datasets across different sequencing platforms.

Dataset		SRR1293902	SRR1293901	GSM1254204	SRR5951587	SRR5951588	SRR5951600
Source		RNA-Seq	RNA-Seq	RNA-Seq	WGS	WGS	WGS
Platform		Illumina HiSeq	Illumina MiSeq	PacBio	Nanopore	Nanopore	Nanopore
Number of Reads		38,278,052	9,524,186	3,239,918	205,685	171,398	161,148
Average Read Length		75	262	1113	3kb	8kb	12kb
Time (Hour)	**TahcoRoll (1-thread)**	1.20	1.40	1.17	0.27	0.44	0.65
	TahcoRoll (8-thread)	0.23	0.28	0.22	0.06	0.09	0.16
	MSBWT	0.95 (4.1X)	1.64 (5.8X)	*NA*	3.03 (53.4X)	2.79 (29.5X)	12.31 (77.7X)
	KMC3	15.85 (68.5X)	14.16 (50.7X)	19.26 (87.3X)	*exceed buffer size*		
	Jellyfish	0.94 (4.0X)	1.56 (5.6X)	1.13 (5.1X)	0.27 (4.8X)	0.48 (5.1X)	0.68 (4.2X)
Memory (GB)	**TahcoRoll**	4.18	4.17	4.18	10.5	10.5	10.5
	MSBWT	7.76 (1.8X)	70.10 (16.8X)	NA	1.89	3.16	4.65
	KMC3	28.76 (6.8X)	24.84 (5.9X)	31.34 (7.4X)	*exceed buffer size*		
	Jellyfish	1.79	1.79	1.79	1.79	1.79	1.79

and {`C`,`T`}. Table 7.7 summarizes the comparison of alternative mappings using real datasets. On average, the default mapping runs faster (shown in Table 7.6) than alternative mappings. Among all balanced partitions, the default setting uses the least amount of memory, but its running time is not significantly different from others (p-values > 0.05). Unbalanced partitions provide more compact representations as revealed by their memory usages, but require more time to resolve collisions than the default setting (p-values < 0.05). The p-values are computed through paired t-tests and adjusted by Bonferroni correction.

Table 7.7: Evaluation of Different Binarized Representations.

Mapping	0={A,C}; 1={G,T}		0={A,T}; 1={C,G}		0={A}; 1={C,G,T}		0={C}; 1={A,G,T}		0={G}; 1 ={A,C,T}		0={T}; 1={A,C,G}	
	Time	**Mem**	**Time**	**Mem**	**Time**	**Mem**	**Time**	**Mem**	**Time**	**Mem**	**Time**	**Mem**
SRR1293902	1.28	4.49	1.27	4.42	1.54	3.36	1.42	3.20	1.54	3.12	1.93	2.98
SRR1293901	1.35	4.49	1.39	4.42	1.72	3.36	1.71	3.20	1.79	3.12	1.96	2.98
GSM1254204	1.26	4.49	1.26	4.42	1.46	3.36	1.64	3.20	1.70	3.12	2.01	2.98
SRR5951587	0.32	11.11	0.30	10.84	0.34	9.85	0.47	7.93	0.57	7.85	0.54	9.27
SRR5951588	0.52	11.11	0.48	10.84	0.58	9.85	0.83	7.93	1.21	7.85	0.81	9.27
SRR5951600	0.76	11.11	0.74	10.84	0.78	9.85	1.46	7.93	1.36	7.85	1.11	9.27
***p*-value**	0.3408		0.12714		0.033552		0.043938		0.008988		0.010212	

7.4 Characterizing Protein-Protein Interaction via Deep Learning

We present the experimental settings and evaluations of PIPR on three PPI prediction tasks, i.e. binary prediction, multi-class interaction type prediction, and binding affinity estimation. The experiments are conducted on three different datasets. Based on the results, we compare and discuss the performance of various baseline approaches.

7.4.1 Binary PPI Prediction

Binary PPI prediction is the primary task targeted by a handful or previous works [26, 84, 87, 101, 103]. The objective of these works is to identify whether a given pair of proteins interacts or not based on their sequences. We use the Yeast benchmark dataset from Guo et al. [25] to compare PIPR with various baseline approaches. In addition, we use the multi-species dataset to demonstrate PIPR's capability of predicting interactions for proteins of different species that share very low sequence identity with those in training.

The baseline approaches include SVM-AC [25], kNN-CTD [101], EELM-PCA [105], SVM-MCD [106], MLP [20], Random Forest LPQ (RF-LPQ) [99], SAE [87], DNN-PPI [45], and DPPI [26]. These approaches vary in terms of the selected features and the classification methods. In addition, we report the results of a Siamese Residual GRU (SRGRU) architecture, which is a simplification of PIPR, where we discard all intermediary convolution layers and keep only the bidirectional residual GRU. The purpose of SRGRU is to show the significance of the contextualized and sequential information of protein profiles in characterizing PPIs. We also report the results of Siamese CNN (SCNN) by removing the residual GRU in PIPR. This degenerates our framework to a similar architecture to DPPI, but differs in that SCNN directly conducts an end-to-end training on raw sequences instead of requiring the protein profiles constructed by PSI-BLAST [3].

We use AMSGrad [74] to optimize the cross-entropy loss, for which we set the learning rate α to 0.001, the exponential decay rates β_1 and β_2 to 0.9 and 0.999, and batch size to 256 on both datasets. The number of occurrences for the RCNN units (i.e., one convolution-pooling layer followed by one bidirectional residual GRU layer) is set to 5, where we adopt 3-max-pooling and the convolution kernel of size 3. We set the hidden state size to be 50, and the RCNN output size to be 100. This configuration ensures the RCNN to compress the selected features in a reasonably small vector sequence before the features are aggregated by the last global average-pooling. We zero-pad short sequences to the longest sequence length in the dataset. This is a widely adopted technique for sequence modeling in NLP [10, 29, 33, 104, 112] as well as in bioinformatics [60, 64, 67] for efficient training. Note that the configuration of embedding pre-training is discussed in Section 7.4.4, and the model configuration study of different hyperparameter values is provided in the Appendix C. All model variants are trained until converge at each fold of the cross-validation.

Following the settings in previous works [26, 84, 87, 103, 106], we conduct 5-fold cross-validation (CV) on the *Yeast dataset*. Under the k-fold CV setting, the data is equally divided into k non-overlapping subsets, and each subset has a chance to train and to test the model so as to ensure an unbiased evaluation. We aggregate fix metrics on the test cases of each fold, i.e. the overall *accuracy*, *precision*, *sensitivity*, *specificity*, *F1*, and *Matthews correlation coefficient (MCC)* on positive cases. All these metrics are preferred to be higher to indicate better performance. Based on the reported accuracy over 5-folds, we also conduct two-tailed Welch's t-tests [97] to evaluate the significance of the improvement on different pairs of approaches. The p-values are adjusted by the Benjamini-Hochberg procedure [6] to control the false discovery rate for multiple hypothesis testing.

As shown in Table 7.8, the CNN-based architecture, DPPI, demonstrates state-of-the-art performance over other baselines that employ statistical learning algorithms or densely connected MLP. This shows the superiority of deep-learning-based techniques in encapsulating various types of information of a protein pair, such as amino acid composition and their co-occurrences, and automatically extracting the robust ones for the learning objectives. That said, DPPI requires an extensive effort in data pre-processing, specifically in constructing the protein profile for each sequence. On average, each PSI-BLAST search of a protein against the NCBI non-redundant protein database (184,243,125 sequences) requires around 90 minutes of computation on our server. Even with eight cores, each search finishes in 15 minutes. We estimate that processing 2,497 sequences of the *Yeast dataset* from scratch can take about 26 days. It is worth mentioning that PIPR only requires 8 seconds to pre-train the amino acid embedding, and 2.5 minutes to train on the *Yeast dataset* (see Table 7.14). We implement SCNN to evaluate the performance of a simplified CNN architecture, which produces comparable results as DPPI. These two frameworks show that CNN can already leverage the significant features from primary protein sequences. In addition, the SRGRU architecture has offered comparable performance to SCNN. This indicates that preserving the sequential and contextualized features of the protein sequences is as crucial as incorporating the local features. By integrating both significant local features and sequential information, PIPR outperforms DPPI by 2.54% in accuracy, 4.93% in sensitivity, and 2.68% in F1-Score. Next, we evaluate whether the improved accuracy of PIPR is statistically significant. Table 7.9 reports the p-values of SRGRU, SCNN, and PIPR compared to other baseline approaches, where the statistically significant comparisons (p-values < 0.01) are highlighted in red. Since the standard deviation of DPPI is unavailable, we are not able to include DPPI in this analysis. The evaluation shows that PIPR performs statistically significantly better than all other approaches, including SCNN and SRGRU. On the other hand, SCNN is not statistically significantly better than SRGRU. Thus, the residual RCNN is very promising for modeling binary PPIs.

Table 7.8: Evaluation of binary PPI prediction on the Yeast dataset.

Methods	**Accuracy (%)**	**Precision (%)**	**Sensitivity (%)**	**Specificity (%)**	**F1-score (%)**	**MCC(%)**
SVM-AC	87.35 ± 1.38	87.82 ± 4.84	87.30 ± 5.23	87.41 ± 6.33	87.34 ± 1.33	75.09 ± 2.51
kNN-CTD	86.15 ± 1.17	90.24 ± 1.34	81.03 ± 1.74	NA	85.39 ± 1.51	NA
EELM-PCA	86.99 ± 0.29	87.59 ± 0.32	86.15 ± 0.43	NA	86.86 ± 0.37	77.36 ± 0.44
SVM-MCD	91.36 ± 0.4	91.94 ± 0.69	90.67 ± 0.77	NA	91.3 ± 0.73	84.21 ± 0.66
MLP	94.43 ± 0.3	96.65 ± 0.59	92.06 ± 0.36	NA	94.3 ± 0.45	88.97 ± 0.62
RF-LPQ	93.92 ± 0.36	96.45 ± 0.45	91.10 ± 0.31	NA	93.7 ± 0.37	88.56 ± 0.63
SAE	67.17 ± 0.62	66.90 ± 1.42	68.06 ± 2.50	66.30 ± 2.27	67.44 ± 1.08	34.39 ± 1.25
DNN-PPI	76.61 ± 0.51	75.1 ± 0.66	79.63 ± 1.34	73.59 ± 1.28	77.29 ± 0.66	53.32 ± 1.05
DPPI	94.55	96.68	92.24	NA	94.41	NA
SRGRU	93.77 ± 0.84	94.60 ± 0.64	92.85 ± 1.58	94.69 ± 0.81	93.71 ± 0.85	87.56 ± 1.67
SCNN	95.03 ± 0.47	95.51 ± 0.77	94.51 ± 1.27	95.55 ± 0.77	95.00 ± 0.50	90.08 ± 0.93
PIPR	**97.09 ± 0.24**	**97.00 ± 0.65**	**97.17 ± 0.44**	97.00 ± 0.67	**97.09 ± 0.23**	**94.17 ± 0.48**

Table 7.9: Statistical assessment on the accuracy of binary PPI prediction.

p-value	**SRGRU**	**SCNN**	**PIPR**
SVM-AC	9.69E-05	1.22E-04	9.69E-05
kNN-CTD	1.03E-05	2.23E-05	2.84E-05
EELM-PCA	2.33E-05	3.94E-08	2.43E-10
SVM-MCD	1.67E-03	2.60E-06	1.35E-07
MLP	1.71E-01	5.29E-02	1.12E-06
RF-LPQ	7.28E-01	4.10E-03	1.75E-06
SAE	4.27E-10	1.78E-10	4.19E-09
DNN-PPI	1.62E-08	2.27E-10	2.70E-09
SRGRU	NA	2.87E-02	6.60E-04
SCNN	2.87E-02	NA	1.80E-04

Table 7.10: Evaluation of binary PPI prediction on variants of multi-species dataset.

Seq. Identity	**# of Proteins**	**Pos. Pairs**	**Neg. Pairs**	**Accuracy (%)**	**F1-Score (%)**
Any	11529	32959	32959	98.19	98.17
<**0.40**	9739	25916	22012	98.29	98.28
<**0.25**	7790	19458	15827	97.91	98.08
<**0.10**	5769	12641	9819	97.54	97.79
<**0.01**	5171	10747	8065	97.51	97.80

We also report the 5-fold CV performance of PIPR on variants of the multi-species dataset, where proteins are excluded based on different thresholds of sequence identity. The dataset contains proteins from *Caenorhabditis elegans*, *Drosophila melanogaster*, and *Escherichia coli*. Table 7.10 shows that PIPR performs consistently well under lenient and stringent criteria of sequence identity between training and testing. More importantly, PIPR is able to train and test on multiple species, and is robust against extremely low sequence identity of less than 1%.

7.4.2 Interaction Type Prediction

The objective of this task is to predict the interaction type of two interacting proteins. We evaluate this task using SHS27k and SHS148k datasets. To the best of our knowledge, fewer efforts attempt for the multi-class PPI prediction in contrast to the binary prediction. Zhu et al. [113] train a two-stage SVM classifier to distinguish obligate, non-obligate, and crystal packing interactions; Silberberg et al. [85] use logistic regression to predict several types of enzymatic actions. However, none of their implementations are publicly available. Different from the categories of interaction types used above, we aim at predicting the interaction types annotated by the STRING database. We train several statistical learning algorithms on the widely employed AC and CTD features for protein characterization as our baselines. These algorithms include SVM, Random Forest, Adaboost (SAMME.R algorithm [27]), kNN classifier, and logistic regression. For deep-learning approaches, we deploy the SCNN architecture where an output MLP with categorical cross-entropy loss is incorporated, as well as a similar SRGRU architecture into comparison. Results of two naïve baselines of random

Table 7.11: Accuracy (%) and fold changes over zero rule for PPI interaction type prediction.

Features	N/A		AC					CTD					Embedded raw seqs		
Methods	Rand	Zero rule	SVM	RF	AdaBoost	kNN	Logistic	SVM	RF	AdaBoost	kNN	Logistic	SCNN	SRGRU	PIPR
SHS27k	14.28	16.70	33.17	44.82	28.67	35.44	25.47	35.56	45.76	31.81	35.56	30.57	55.54	51.06	**59.56**
(fold×)	—	1.00×	1.99×	2.68×	1.72×	2.12×	1.52×	2.13×	2.74×	1.90×	2.13×	1.83×	3.33×	3.06×	**3.57**×
SHS148k	14.28	16.21	28.17	36.01	27.87	33.81	24.96	31.37	36.65	29.67	33.13	26.96	55.29	54.05	**61.91**
(fold×)	—	1.00×	1.74×	2.22×	1.72×	2.09×	1.54×	1.94×	2.26×	1.83×	2.04×	1.66×	3.41×	3.33×	**3.82**×

guessing and zero rule (i.e., predicting the majority class) are also reported for reference.

All approaches are evaluated on the two datasets by 10-fold CV, using the same partition scheme for a more unbiased evaluation [34, 56]. We carry forward the model configurations from the last experiment to evaluate the performance of the frameworks under controlled variables. For baseline models, we examine three different ways of combining the feature vectors of the two input proteins, i.e. element-wise multiplication, the Manhattan difference (i.e. the absolute differences of corresponding features [64]) and concatenation. The Manhattan difference consistently obtains better performance, considering the small values of the input features and the asymmetry of the captured protein relations.

The prediction accuracy and fold changes over the zero rule baseline are reported in Table 7.11. Note that since the multi-class prediction task is much more challenging than the binary prediction task, it is expected to observe lower accuracy and longer training-time (Table 7.14) than that reported in the previous experiment. Among all the baselines using explicit features, the CTD-based models perform better than the AC-based ones. CTD descriptors seek to cover both continuous and discontinuous interaction information [101], which potentially better discriminate among PPI types.

The best baseline using Random Forest thereof achieves satisfactory results by more than doubling the accuracy of zero rule on the smaller SHS27k dataset. However, on the larger SHS148k dataset, the accuracy of these explicit-feature-based models is notably impaired. We hypothesize that such predefined explicit features are not representative enough to distinguish the PPI types. On the other hand, the deep-learning-based approaches do not need to explicitly utilize these features, and perform consistently well in both settings. The raw sequence information is sufficient for these approaches to drastically outperform the Random Forest by at least 5.30% in accuracy on SHS27k and 17.40% in accuracy on

SHS148k. SCNN thereof outperforms SRGRU by 4.48% and 1.24% in accuracy on SHS27k and SHS148k, respectively. This implies that the local interacting features are relatively more deterministic than contextualized and sequential features on this task. The results by the residual RCNN-based framework are very promising, as it outperforms SCNN by 4.02% and 6.62% in accuracy on SHS27k and SHS148k respectively. It also remarkably outperforms the best explicit-feature-based baselines on the two datasets by 13.80% and 25.26% in accuracy, and more than 3.5 of fold changes over the zero rule on both datasets.

7.4.3 Binding Affinity Estimation

As the last task, we evaluate PIPR for binding affinity estimation using the SKEMPI dataset. We employ the mean squared loss variant of PIPR to address this regression task. Since the lengths of protein sequences in SKEMPI are much shorter than those in the other datasets, we accordingly reduce the occurrences of RCNN units to 3, while other configurations remain unchanged. For baselines, we compare against several regression models based on the AC and CTD features, which include Bayesian Redge regressor (BR), SVM, Adaboost with decision tree regressors and Random Forest regressor. The corresponding features for two sequences are again combined via the Manhattan difference. We also modify SCNN and SRGRU to their mean squared loss variants, in which we reduce the layers in the same way of RCNN.

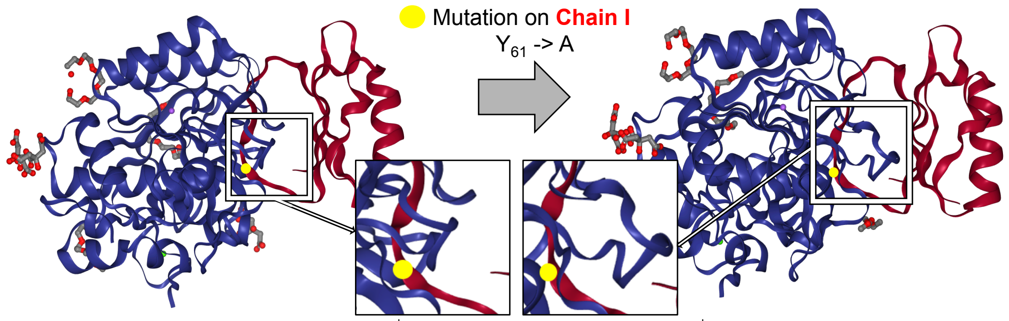
We aggregate three metrics through 10-fold CV, i.e. *mean squared error* (*MSE*), *mean absolute error* (*MAE*) and *Pearson's correlation coefficient* (*Corr*). These are commonly reported metrics for regression tasks, for which lower *MSE* and *MAE* as well as higher *Corr* indicate better performance. In the cross-validation process, we normalize the affinity values of the SKEMPI dataset to $[0, 1]$ via min-max re-scaling. This is due the fact that we use sigmoid function to smooth the output of the regressor. Note that it does not affect correlation, while MSE, MAE and the original affinity scores can be easily re-scaled back.

Table 7.12 reports the results for this experiment. It is noteworthy that, one single change of amino acid can lead to a drastic effect on binding affinity. While such subtle changes are difficult to be reflected by the explicit features, the deep-learning-based methods can com-

Table 7.12: Evaluation of binding affinity prediction on the SKEMPI dataset.

Features	AC				CTD				Embedded raw seqs		
Methods	BR	SVM	RF	AdaBoost	BR	SVM	RF	AdaBoost	SCNN	SRGRU	PIPR
$MSE(\times 10^{-2})$	1.70	2.20	1.77	1.98	1.86	1.84	1.49	1.84	0.87	0.95	**0.63**
$MAE(\times 10^{-2})$	9.56	11.81	9.81	11.15	10.20	11.04	9.06	10.69	6.49	7.08	**5.48**
Corr	0.564	0.353	0.546	0.451	0.501	0.501	0.640	0.508	0.831	0.812	**0.873**

petently capture such changes from the raw sequences. Our RCNN-based framework again offers the best performance among the deep-learning-based approaches, and significantly outperforms the best baseline (CTD-based Random Forest) by offering a 0.233 increase in *Corr*, as well as remarkably lower *MSE* and *MAE*. Figure 7.15 demonstrates an example of the effect of changing an amino acid in a protein complex. The blue entity is Subtilisin BPN' precursor (Chain E), and the red entity is Chymotrypsin inhibitor (Chain I). The mutation is highlighted in yellow. The wild type (1TM1) and mutant (1TO1) complexes are retrieved from PDB. Tyrosine at position 61 of Chymotrypsin inhibitor 2 (Chain I) is substituted with Alanine, causing the neighboring region of Subtilisin BPN' precursor (Chain E) to relax. The binding affinity (k_d) changes from 2.24E-12 to 2.70E-10, which is validly captured by PIPR. While our experiment is conducted on a relatively small dataset, we seek to extend our PIPR framework to a more generalized solution for binding affinity estimation, once a larger and more heterogeneous corpus is available.



PDB ID	**1TM1 (wild type)**	**1TO1 (mutant)**
Binding Affinity (k_d)	2.24E-12	2.70E-10
PIPR Prediction (k_d)	2.54E-12	3.52E-10

Figure 7.15: Mutation effects on structure and binding affinity.

Table 7.13: Comparison of amino acid representations based on binary prediction.

	$[\mathbf{a}_c, \mathbf{a}_{ph}]$	$\mathbf{a}_c$ only	$\mathbf{a}_{ph}$ only	One-hot
Dimension	12	5	7	20
Accuracy	**97.09**	96.67	96.03	96.11
Precision	**97.00**	96.35	95.91	96.34
F1-Score	**97.09**	96.51	96.08	96.10

7.4.4 Amino Acid Embeddings

Each amino acid is represented by a vector of numerical values that describe its relative physicochemical properties. The first part of the embedding vector $\mathbf{a}_c$, which measures the co-occurrence similarity of the amino acids in protein sequences, is empirically set as a 5-dimensional vector. $\mathbf{a}_c$ is obtained by pre-training the Skip-Gram model on all 8,000 sequences from our largest STRING dataset, SHS148k, using a context window size of 7 and a negative sampling size of 5. The second part contains a 7-dimensional vector, $\mathbf{a}_{ph}$, which describes the categorization of electrostaticity and hydrophobicity for the amino acid. We examine the performance of using each part individually, as well as the performance of combining them as used in our framework. In addition, we include a naïve one-hot vector representation, which does not consider the relatedness of amino acids and treats each of them independently. Table 7.13 shows that, once we remove either of the two parts of the proposed embedding, the performance of the model slightly drops. Meanwhile, the proposed pre-trained embeddings lead to noticeably better performance of the model than adopting the naïve one-hot encodings of the canonical amino acids. This pre-training process completes in 8 seconds on a commodity workstation as shown in Table 7.14. This is a one-time effort that can be reused on different tasks and datasets.

7.4.5 Run-time Analysis

All of the experiments are conducted on one NVIDIA GeForce GTX 1080 Ti GPU. We report the training time for each experiment, as well as for the amino acid embedding in Table 7.14. For each experiment, we calculate the average training time over either 5-

Table 7.14: Run-Time of training embeddings and different prediction tasks.

Task	**Embeddings**	**Binary**	**Multi-class**	**Multi-class**	**Regression**
Dataset	**SHS148k**	**Yeast**	**SHS27k**	**SHS148k**	**SKEMPI**
Sample Size	8,000	11,188	26,945	148,051	2,950
Training Time	8sec	2.5min	15.8min	138.3min	12.5min

fold (Yeast dataset) or 10-fold (others) CV. In both binary and multi-class predictions, the training time increases along with the increased number of training cases. The regression estimation generally requires more iterations per training case to converge than classification tasks. Thus, with much fewer cases, the training time on SKEMPI for affinity estimation is more than that on the Yeast dataset for binary prediction.

CHAPTER 8

Conclusion

We have introduced three computational frameworks to address the challenges in leveraging sequence information to extract biological knowledge. The first framework aims at correcting the alignment of short sequences generated from genes located in the regions with low sequence complexity. The corrected alignments further facilitate accurate expression estimates among homologous genes. The second framework, Fleximer, contains a series of algorithms and statistical methods to quantify transcript expression abundance via a set of attentively selected *k-mers* with variable lengths. The third framework, PIPR, employs a Siamese architecture to capture the mutual influence of a protein sequence pair in characterizing the interaction properties. The contributions of this dissertation in advancing the sequence analysis of genomic and proteomic data can be summarized as follows:

- We present a genome-wide approach to correct read alignment among homologous loci, specifically for pseudogenes and their parents. Our approach directly models the alignment distribution among homologous loci, and correct the falsely aligned reads.

- To expedite the analyses, we incorporate community detection algorithm to partition the genome into different homologous communities, and build a linear regression model separately for each community. The non-negative least square problem is then solved on a smaller dataset with a further potential of parallelism.

- We emphasize the importance of considering a list of *k-mers* with different sizes to quantify the transcript abundance via RNA-Seq.

- To overcome the limitation of a fixed k in transcript abundance quantification, we develop an efficient approach that leverages the structure of suffix tree and the concept of splicing graph to discover and select an optimal set of variable-length *k-mers*. The quantification step relies on the Aho-Corasick algorithm and the EM algorithm for relative abundance estimations.

- Further accelerating the matching process between variable length *k-mers* and sequencing reads, we propose a thinned Aho-Corasick algorithm which incorporates the one-bit representation for four nucleotides and a rolling hash technique for a loss-less matching.
- We construct an end-to-end framework for PPI prediction that requires only the primary protein sequences as the input. It is trained to automatically preserve the critical features from the sequences.
- When analyzing the protein sequences, we emphasize and demonstrate the needs of considering the contextualized and sequential information in the model.
- The proposed framework can be flexibly used to address different PPI tasks, including binary interaction prediction, multi-class interaction type prediction, and binding affinity estimations.

Each framework contains several components to address specific aspect of the challenges. We have also demonstrated the effectiveness in terms of accuracy and/or efficiency using a wide range of datasets in the different experiments.

The algorithmic techniques developed in Fleximer can be easily extended for tasks that involve processing vast sequencing data. These application includes, but are not limited to metagenomic sequencing, bisulfite sequencing, ChIPseq, miRNASeq and cfDNA-Seq (cell free DNA sequencing). Specifically, the framework of Fleximer can be applied to metagenomic studies. A set of variable-length signatures bears more discriminating power in distinguishing reads among strains, and thus provides the potential of quantifying the microbial abundance down to the strain level.

The framework of PIPR provides an automatic multi-granular feature selection mechanism to capture both local significant features and sequential features from the protein sequences. One important direction is to apply the PIPR framework to other sequence-based inference tasks in bioinformatics, such as modeling RNA and protein interactions. We also seek to incorporate attention mechanisms [94] to help pinpoint interaction sites on protein sequences. Since PIPR has alleviated any costly domain-invariant feature engineering process, how to extend PIPR with transfer learning based domain adaptation for different species is another meaningful direction.

APPENDIX A

Background on Suffix Tree

Let t' be a string over the alphabet $\Sigma = \{A, C, G, T\}$, and "\$" be a unique character, such that $\$ \notin \Sigma$. This unique character is appended to the end of the string t' to guarantee that every suffix ends at a leaf in the tree. Therefore, we use the following notations: t denotes the entire string with a terminator ($t' + \$$), and $|t|$ denotes the size of t. The string t can be degenerated into $|t|$ suffixes, and stored into a compressed tree structure, known as the suffix tree. This suffix tree contains exactly $|t|$ leaves, which are labeled by the starting position of the corresponding suffix. Each internal node has at least two children. The label of an internal node is omitted for memory efficiency. Each edge is labeled with a non-empty substring of t; edges connecting from the same parent contains different starting characters. The edge labels can be concatenated through a path in the tree, forming a longer substring of t. Thus, any individual suffix of t can be recreated by traversing along a path from the root to a leaf and concatenating the labels of the edges along the way.

Multiple sequences can be stored in the same tree, forming a generalized suffix tree (GST) [8]. A GST can be constructed by appending different unique symbols at the end of each sequence to ensure no suffix is a substring of another. Alternatively, we can concatenate sequences into one long string, separated by a single terminator, and build a regular suffix tree. Regardless of using a single or multiple terminators, each leaf node stores sufficient information to retrieve the starting positions and sequence of origin for each suffix.

The construction of a suffix tree takes linear time and space respect to the total sequence size [21, 55, 92]. The most memory efficient implementation to handle genomic sequences is the Sadakane's compressed suffix tree [80, 93], which relies on several abstract data structures to reduce the memory footprint. It also preserves all the operations for a normal suffix tree. One of the abstract data structures is balanced parentheses [63], which represents the tree in pre-order. This structure can be further modified to retrieve node information in post-order. For these reasons, we implement our method using Sadakane's compressed suffix tree, which is available from Succinct Data Structure Library (SDSL) [1].

[1]https://github.com/simongog/sdsl-lite/

APPENDIX B

Propositions in TahcoRoll

B.1 Proof of Proposition 1

Proof. Given the prefix length i and the number of possible characters c, there are c^i possible prefixes in total. Assuming the characters are uniformly distributed, the probability that a particular prefix exists in n *sig-mers* is:

$$1-\left(1-\frac{1}{c^i}\right)^n.$$

Therefore, the expected number of collided prefixes is:

$$n-c^i\left(1-\left(1-\frac{1}{c^i}\right)^n\right),$$

and the expected number of prefixes without any collision is:

$$n-\left(n-c^i\left(1-\left(1-\frac{1}{c^i}\right)^n\right)\right)=c^i\left(1-\left(1-\frac{1}{c^i}\right)^n\right)=c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right).$$

However, the expected number above includes the cases that fail before reaching the i-th character. Hence, the expected number of these cases should be deducted. Finally, the expected number of *sig-mers* that fail to find their length-i prefixes along the trie during its insertion is:

$$c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right)-c^{i-1}\left(1-\left(\frac{c^{i-1}-1}{c^{i-1}}\right)^n\right).$$

□

B.2 Proof of Proposition 2

From Proposition 1, the expected number of node for prefix length i in n *sig-mers* is $c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right)$. Intuitively, summing all possible prefix lengths up to the length of *sig-mer* m, the expected number of trie nodes is

$$\sum_{i=1}^{m} c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right).$$

Proof. Denote the expected number of *sig-mers* that fail to find their length-i prefixes on the trie during its insertion as $f(i)$. Given the length of *sig-mers* m, each *sig-mer* that fails to find the length-i prefix along the trie during its insertion will result in the addition of $m-i+1$ nodes. Based on Proposition 1, the expected number of nodes in the trie is:

$$\begin{aligned}
\sum_{i=1}^{m}(m-i+1)\cdot f(i) &= \sum_{i=1}^{m}(m-i+1)\left[c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right)-c^{i-1}\left(1-\left(\frac{c^{i-1}-1}{c^{i-1}}\right)^n\right)\right]\\
&= \sum_{i=1}^{m}(m-i+1)\left[c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right)\right]-\sum_{i=1}^{m-1}(m-i)\left[c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right)\right]\\
&= \sum_{i=1}^{m-1}\left[(m-i+1)-(m-i)\right]\left[c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right)\right]+c^m\left(1-\left(\frac{c^m-1}{c^m}\right)^n\right)\\
&= \sum_{i=1}^{m} c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right).
\end{aligned}$$

□

B.3 Proof of Proposition 3

Proof. Suppose that the number of *sig-mers* to be added into a trie is extremely large. The expected number of nodes with c possible characters is:

$$\lim_{n\to\infty}\sum_{i=1}^{m} c^i\left(1-\left(\frac{c^i-1}{c^i}\right)^n\right)=\sum_{i=1}^{m} c^i=\frac{c\left(c^m-1\right)}{c-1}=\frac{c^{m+1}-c}{c-1}.$$

For the plain AC, there are four possible characters, i.e., `A`, `C`, `G` and `T`. Hence, the expected number of its nodes N_A is:

$$N_A = \frac{4^{m+1} - 4}{4 - 1} = \frac{(2^2)^{m+1} - 4}{3} = \frac{2^{2m+2} - 4}{3} = \frac{4\,(2^{2m} - 1)}{3}.$$

For the thinned automaton, there are two possible characters, i.e., `0` and `1`. Hence, the expected number of its nodes N_T is:

$$N_T = \frac{2^{m+1} - 2}{2 - 1} = 2(2^m - 1).$$

Finally, we compute the ratio of the expected number of two approaches as follows:

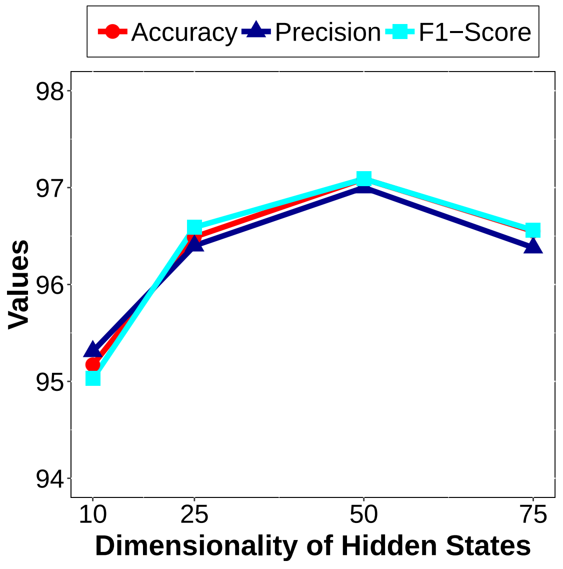
$$\frac{N_T}{N_A} = \frac{2(2^m - 1)}{\frac{4(2^{2m}-1)}{3}} = \frac{3}{2} \cdot \frac{2^m - 1}{2^{2m} - 1} = \frac{3}{2} \cdot \frac{2^m - 1}{(2^m + 1)\,(2^m - 1)} = \frac{3}{2} \cdot \frac{1}{2^m + 1}.$$

$\square$

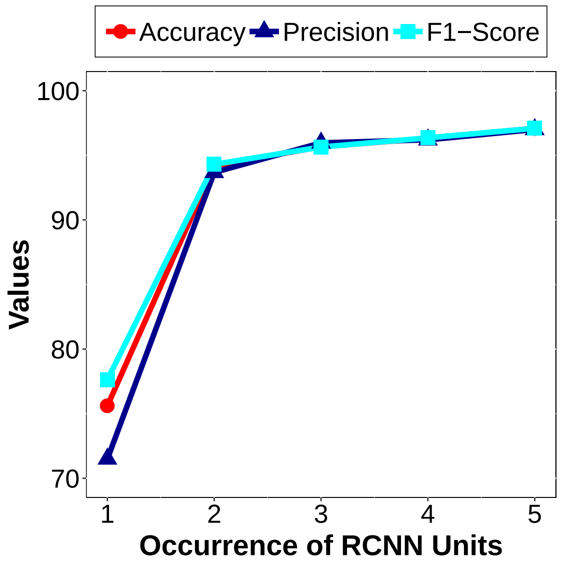
APPENDIX C

Hyperparameter Study for PIPR

We examine the configuration of two critical factors that can affect the performance of PIPR: the dimensionality of hidden states and the number of occurrences for the RCNN units. We show the effects of different settings of these two factors based on the binary PPI prediction task. The hidden state sizes are chosen from {10, 25, 50, 75}. As illustrated in Fig C.1a, the performance of PIPR initially increases as we raise the dimensionality of the hidden states until it passes 50, and then starts to decline. The occurrences of RCNN units contribute to the levels of granularity in feature aggregation. Fewer occurrences correspond to less aggregation. However, too many occurrences can lead to over-compressing the features. We examine the occurrences from 1 to 5 based on Yeast. Note that we do not adopt the setting with 6 occurrences, where the RCNN encoder over-compresses the extracted features to a very small number of latent vectors before the last global average pooling. Aligned with our hypothesis, Fig C.1b shows that the accuracy, precision, and F1-score improve when we increase the number of occurrences of the RCNN units. The improvement from 2 to 5 occurrences is marginal, which shows that our framework is robust to this setting as long as there are more than 2 occurrences of RCNN units.



(a) Hidden states.



(b) RCNN units.

Figure C.1: Hyperparameter Study for PIPR.

APPENDIX D

Software Configuration for *K-mer* Counters

Existing *k-mer* counters index the reads into a compact and searchable structure, such as a hash table, a burst trie, or a compact suffix array. The occurrences of a specific k-mer can be retrieved by querying these data structures. Most of these counters are designed to process reads with fixed-size k-mers; several of them restrict the choice of k to fall within a threshold. These algorithms can be adapted to count k-mers of different sizes by repeating the process with different k's. Here, we discuss these approaches.

D.1 Thread-Safe Shared Memory Hashing

Jellyfish [54] exploits the CAS (compare-and-swap) assembly instruction to update a memory location in a multi-threaded environment, and uses the 'quotienting technique' and bit-packed data structure to reduce wasted memory. It provides a function (`--if=kmerfile`) to count only a list of specific k-mers with the same k. The counting step is repeated for different k. **Squeakr** [68] builds an off-the-shelf data structure based on counting quotient filter (CQF). It maintains both global and local CQFs to facilitate updates of each thread.

D.2 Disk-Based Hashing

Disk-based hashing reduces memory with complementary disk space. In general, this approach splits k-mers into bins, and stores in files. Each bin is then loaded back for counting.

DSK [75] divides k-mers using a specific hash function based on the targeted memory and disk space. To account for arbitrary k-mer lengths, we compile the source code with default parameters (`-DKSIZE_LIST=32`). For each experiment, we first load the k-mers into memory and determine the range of the k-mer sizes. We index reads with different k's using the main program, and dump the result into a human readable format with `dsk2ascii`.

MSPKmerCounter (MSPKC) [49] proposes Minimum Substring Partitioning to reduce the memory usage of storing k-mers. Observing the fact that consecutive k-mers in a read often share a shorter substring, these consecutive k-mers can be compressed and stored in one bin. It is recommended to index reads with an odd number k less than 64. The software contains three functions to be run in sequence: `partition`, `count`, and `dump`. The partition step divides data using minimum substring partitioning, the count step computes the frequencies of existing k-mers, and the dump step converts the results into human readable format. This sequel is repeated for each k.

KMC [17], **KMC2** [18], and **KMC3** [41] are serial developments of parallel counters. These methods scan reads one block at a time and use a number of splitter threads to process the blocks. KMC2 leverages the concept of minimizer to further reduce disk usage. KMC3 accelerates the running time and optimizes the memory by taking a larger part of input data and better balancing the bin sizes. We use the main program from KMC3 to count the k-mer of all sizes seen in the list, with one k at a time. We set the `-cs` parameter to 4294967295 to ensure all of the frequently occurred *k-mers* are included.

D.3 Probabilistic Hashing

This approach avoids counting the k-mers that are likely to arise from sequencing errors.

BFCounter [58] uses Bloom filter to identify all k-mers that are present more frequently than a threshold with a low false positive rate. The algorithm scans read data in two passes. The count function of BFCounter requires an estimation of the number of *k-mers*. We use *KmerStream* [57] to pre-compute the *k-mer* statistics in reads. We use the `dump` function to convert the results into human readable format to extract the frequencies.

khmer [108] uses a streaming-based probabilistic data structure, CountMin Sketch [16].

The algorithm is designed to perform in-memory counting, and cannot handle k larger than 32. We use its Python wrapper script, `load-into-counting`, to perform counting, which writes a k-mer graph for each k to files. We repeat this step for different k's. Each k-mer graph is loaded back to the memory one at a time, allowing us to query the count. We set the maximum amount of memory for data structure to be 16G as the required parameter.

D.4 Suffix-Arrays

Suffix-arrays present the potential of searching arbitrary k-mers on a single scan. However, constructing a suffix-array on read data can be computationally expensive.

Tallymer [43] is tailored to detect *de novo* repetitive elements ranging from 10 to 500bp in the genome. The algorithm first constructs an enhanced suffix array (`gt suffixerator`), and indexes k-mers one k at a time. We use `gt tallymer mkindex` to extract the k-mer index from the enhanced suffix-array, and use `gt tallymer search` to retrieve their counts.

MSBWT [32] compresses raw reads via a multi-string variant of Burrows-Wheeler Transform (BWT). Instead of concatenating all reads and sorting, it builds a BWT on each string and merges these multi-string BWTs through a small interleave array. The final structure allows a fast query of k-mers of arbitrary k.

D.5 Burst Tries

KCMBT [53] uses a cache efficient burst trie to store compact k-mers. The trie structure stores k-mers that share the same prefix in the same container. When a container is full, k-mers are sorted and burst. A good balance between the container size and the tree depth is essential to avoid constant sorting and bursting. As a result, KCMBT uses hundreds of trees. Unfortunately, it is limited to process k-mers with k less than 32. We first load the list of k-mers into memory. KCMBT generates binary files containing *k-mers* and their counts. We use `kcmbt_dump` to convert the binary data into human readable files.

BIBLIOGRAPHY

[1] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975. 4, 17, 22

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990. 15

[3] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped blast and psi-blast: a new generation of protein database search programs," *Nucleic acids research*, vol. 25, no. 17, pp. 3389–3402, 1997. 8, 62

[4] K. F. Au, H. Jiang, L. Lin, Y. Xing, and W. H. Wong, "Detection of splice junctions from paired-end rna-seq data by splicemap," *Nucleic acids research*, vol. 38, no. 14, pp. 4570–4578, 2010. 5

[5] K. F. Au, V. Sebastiano, P. T. Afshar, J. D. Durruthy, L. Lee, B. A. Williams, H. van Bakel, E. E. Schadt, R. A. Reijo-Pera, J. G. Underwood *et al.*, "Characterization of the human esc transcriptome by hybrid sequencing," *Proceedings of the National Academy of Sciences*, vol. 110, no. 50, pp. E4821–E4830, 2013. 39

[6] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: a practical and powerful approach to multiple testing," *Journal of the Royal statistical society: series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995. 63

[7] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, "The protein data bank," *Nucleic acids research*, vol. 28, no. 1, pp. 235–242, 2000. 41

[8] P. Bieganski, J. Riedl, J. V. Carlis, and E. F. Retzel, "Generalized suffix trees for biological sequence data: Applications and implementation," in *HICSS (5)*, 1994, pp. 35–44. 4, 74

[9] N. L. Bray, H. Pimentel, P. Melsted, and L. Pachter, “Near-optimal probabilistic rna-seq quantification,” *Nature biotechnology*, vol. 34, no. 5, p. 525, 2016. 6, 28, 49

[10] M. Chen, C. Meng, G. Huang, and C. Zaniolo, “Neural article pair modeling for wikipedia sub-article matching,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer, 2018, pp. 3–19. 32, 62

[11] H. Cho, J. Davis, X. Li, K. S. Smith, A. Battle, and S. B. Montgomery, “High-resolution transcriptome analysis with long-read rna sequencing,” *PLoS One*, vol. 9, no. 9, p. e108095, 2014. 39

[12] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014. 33

[13] J. D. Cohen, “Recursive hashing functions for n-grams,” *ACM Transactions on Information Systems (TOIS)*, vol. 15, no. 3, pp. 291–320, 1997. 26

[14] A. Conneau, H. Schwenk, L. Barrault, and Y. Lecun, “Very deep convolutional networks for text classification,” *arXiv preprint arXiv:1606.01781*, 2016. 34

[15] U. Consortium *et al.*, “Uniprot: the universal protein knowledgebase,” *Nucleic acids research*, vol. 46, no. 5, p. 2699, 2018. 40

[16] G. Cormode and S. Muthukrishnan, “An improved data stream summary: the count-min sketch and its applications,” *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005. 82

[17] S. Deorowicz, A. Debudaj-Grabysz, and S. Grabowski, “Disk-based k-mer counting on a pc,” *BMC bioinformatics*, vol. 14, no. 1, p. 160, 2013. 7, 82

[18] S. Deorowicz, M. Kokot, S. Grabowski, and A. Debudaj-Grabysz, “Kmc 2: fast and resource-frugal k-mer counting,” *Bioinformatics*, vol. 31, no. 10, pp. 1569–1576, 2015. 7, 82

[19] B. Dhingra, H. Liu, Z. Yang, W. W. Cohen, and R. Salakhutdinov, “Gated-attention readers for text comprehension,” *arXiv preprint arXiv:1606.01549*, 2016. 33

[20] X. Du, S. Sun, C. Hu, Y. Yao, Y. Yan, and Y. Zhang, “Deepppi: boosting prediction of protein–protein interactions with deep neural networks,” *Journal of chemical information and modeling*, vol. 57, no. 6, pp. 1499–1510, 2017. 8, 62

[21] M. Farach, “Optimal suffix tree construction with large alphabets,” in *Proceedings 38th Annual Symposium on Foundations of Computer Science*. IEEE, 1997, pp. 137–143. 74

[22] A. C. Frazee, A. E. Jaffe, B. Langmead, and J. T. Leek, “Polyester: simulating rna-seq datasets with differential transcript expression,” *Bioinformatics*, vol. 31, no. 17, pp. 2778–2784, 2015. 38

[23] M. Garber, M. G. Grabherr, M. Guttman, and C. Trapnell, “Computational methods for transcriptome annotation and quantification using rna-seq,” *Nature methods*, vol. 8, no. 6, p. 469, 2011. 2, 5

[24] G. H. Gonnet and R. A. Baeza-Yates, “An analysis of the karp-rabin string matching algorithm,” *Information Processing Letters*, vol. 34, no. 5, pp. 271–274, 1990. 27

[25] Y. Guo, L. Yu, Z. Wen, and M. Li, “Using support vector machine combined with auto covariance to predict protein–protein interactions from protein sequences,” *Nucleic acids research*, vol. 36, no. 9, pp. 3025–3030, 2008. 8, 40, 62

[26] S. Hashemifar, B. Neyshabur, A. A. Khan, and J. Xu, “Predicting protein–protein interactions through sequence-based deep learning,” *Bioinformatics*, vol. 34, no. 17, pp. i802–i810, 2018. 8, 32, 36, 40, 62, 63

[27] T. Hastie, S. Rosset, J. Zhu, and H. Zou, “Multi-class adaboost,” *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009. 65

[28] P. G. Hawkins and K. V. Morris, "Transcriptional regulation of oct4 by a long non-coding rna antisense to oct4-pseudogene 5," *Transcription*, vol. 1, no. 3, pp. 165–175, 2010. 6

[29] H. He, K. Gimpel, and J. Lin, "Multi-perspective sentence similarity modeling with convolutional neural networks," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1576–1586. 34, 62

[30] S. Heber, M. Alekseyev, S.-H. Sze, H. Tang, and P. A. Pevzner, "Splicing graphs and est assembly problem," *Bioinformatics*, vol. 18, no. suppl_1, pp. S181–S188, 2002. 19

[31] M. Höhl, S. Kurtz, and E. Ohlebusch, "Efficient multiple genome alignment," *Bioinformatics*, vol. 18, no. suppl_1, pp. S312–S320, 2002. 17

[32] J. Holt and L. McMillan, "Merging of multi-string bwts with applications," *Bioinformatics*, vol. 30, no. 24, pp. 3524–3531, 2014. 7, 83

[33] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in *Advances in neural information processing systems*, 2014, pp. 2042–2050. 62

[34] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112. 66

[35] J.-Y. Jiang, F. Chen, Y.-Y. Chen, and W. Wang, "Learning to disentangle interleaved conversational threads with a siamese hierarchical network and similarity ranking," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018, pp. 1812–1822. 36

[36] L. Kaderali and A. Schliep, "Selecting signature oligonucleotides to identify organisms using dna arrays," *Bioinformatics*, vol. 18, no. 10, pp. 1340–1349, 2002. 17

[37] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM journal of research and development*, vol. 31, no. 2, pp. 249–260, 1987. 22, 26

[38] 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*, vol. 14, no. 4, p. R36, 2013. 5, 42

[39] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1646–1654. 34

[40] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014. 32

[41] M. Kokot, M. Długosz, and S. Deorowicz, "Kmc 3: counting and manipulating k-mer statistics," *Bioinformatics*, vol. 33, no. 17, pp. 2759–2761, 2017. 7, 82

[42] S. Kurtz and C. Schleiermacher, "Reputer: fast computation of maximal repeats in complete genomes." *Bioinformatics (Oxford, England)*, vol. 15, no. 5, pp. 426–427, 1999. 17

[43] S. Kurtz, A. Narechania, J. C. Stein, and D. Ware, "A new method to compute k-mer frequencies and its application to annotate large repetitive plant genomes," *BMC genomics*, vol. 9, no. 1, p. 517, 2008. 7, 83

[44] B. Li and C. N. Dewey, "Rsem: accurate transcript quantification from rna-seq data with or without a reference genome," *BMC bioinformatics*, vol. 12, no. 1, p. 323, 2011. 5, 6

[45] H. Li, X.-J. Gong, H. Yu, and C. Zhou, "Deep neural network based predictions of protein interactions using primary sequences," *Molecules*, vol. 23, no. 8, p. 1923, 2018. 8, 62

[46] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, “The sequence alignment/map format and samtools,” *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009. 14

[47] M. Li, I. X. Wang, Y. Li, A. Bruzel, A. L. Richards, J. M. Toung, and V. G. Cheung, “Widespread rna and dna sequence differences in the human transcriptome,” *science*, vol. 333, no. 6038, pp. 53–58, 2011. 5

[48] W. Li and A. Godzik, “Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences,” *Bioinformatics*, vol. 22, no. 13, pp. 1658–1659, 2006. 40

[49] Y. Li *et al.*, “Mspkmercounter: a fast and memory efficient approach for k-mer counting,” *arXiv preprint arXiv:1505.06550*, 2015. 7, 82

[50] H. Lin, A. Shabbir, M. Molnar, and T. Lee, “Stem cell regulatory function mediated by expression of a novel mouse oct4 pseudogene,” *Biochemical and biophysical research communications*, vol. 355, no. 1, pp. 111–116, 2007. 6

[51] M. Lin, Q. Chen, and S. Yan, “Network in network,” *arXiv preprint arXiv:1312.4400*, 2013. 35

[52] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models.” 36

[53] A.-A. Mamun, S. Pal, and S. Rajasekaran, “Kcmbt: ak-mer counter based on multiple burst trees,” *Bioinformatics*, vol. 32, no. 18, pp. 2783–2790, 2016. 7, 83

[54] G. Marçais and C. Kingsford, “A fast, lock-free approach for efficient parallel counting of occurrences of k-mers,” *Bioinformatics*, vol. 27, no. 6, pp. 764–770, 2011. 7, 81

[55] E. M. McCreight, “A space-economical suffix tree construction algorithm,” *Journal of the ACM (JACM)*, vol. 23, no. 2, pp. 262–272, 1976. 74

[56] G. McLachlan, K.-A. Do, and C. Ambroise, *Analyzing microarray gene expression data.* John Wiley & Sons, 2005, vol. 422. 66

[57] P. Melsted and B. V. Halldórsson, “Kmerstream: streaming algorithms for k-mer abundance estimation,” *Bioinformatics*, vol. 30, no. 24, pp. 3541–3547, 2014. 82

[58] P. Melsted and J. K. Pritchard, “Efficient counting of k-mers in dna sequences using a bloom filter,” *BMC bioinformatics*, vol. 12, no. 1, p. 333, 2011. 7, 82

[59] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in *Advances in neural information processing systems*, 2013, pp. 3111–3119. 35

[60] X. Min, W. Zeng, N. Chen, T. Chen, and R. Jiang, “Chromatin accessibility prediction via convolutional long short-term memory networks with k-mer embedding,” *Bioinformatics*, vol. 33, no. 14, pp. i92–i101, 2017. 62

[61] I. H. Moal and J. Fernández-Recio, “Skempi: a structural kinetic and energetic database of mutant protein interactions and its use in empirical models,” *Bioinformatics*, vol. 28, no. 20, pp. 2600–2607, 2012. 41

[62] A. Mortazavi, B. A. Williams, K. McCue, L. Schaeffer, and B. Wold, “Mapping and quantifying mammalian transcriptomes by rna-seq,” *Nature methods*, vol. 5, no. 7, p. 621, 2008. 6

[63] J. I. Munro and V. Raman, “Succinct representation of balanced parentheses and static trees,” *SIAM Journal on Computing*, vol. 31, no. 3, pp. 762–776, 2001. 75

[64] A. T. Muller, J. A. Hiss, and G. Schneider, “Recurrent neural network model for constructive peptide design,” *Journal of chemical information and modeling*, vol. 58, no. 2, pp. 472–479, 2018. 62, 66

[65] G. Navarro and M. Raffinot, *Flexible pattern matching in strings: practical on-line*

search algorithms for texts and biological sequences. Cambridge University Press, 2002. 23

[66] L. Pachter, "Models for transcript quantification from rna-seq," *arXiv preprint arXiv:1104.3889*, 2011. 5, 17

[67] X. Pan and H.-B. Shen, "Predicting rna–protein binding sites and motifs through combining local and global deep convolutional neural networks," *Bioinformatics*, vol. 34, no. 20, pp. 3427–3436, 2018. 62

[68] P. Pandey, M. A. Bender, R. Johnson, and R. Patro, "Squeakr: an exact and approximate k-mer counting system," *Bioinformatics*, vol. 34, no. 4, pp. 568–575, 2017. 7, 81

[69] B. Paşaniuc, N. Zaitlen, and E. Halperin, "Accurate estimation of expression levels of homologous genes in rna-seq experiments," *Journal of Computational Biology*, vol. 18, no. 3, pp. 459–468, 2011. 6

[70] R. Patro, S. M. Mount, and C. Kingsford, "Sailfish enables alignment-free isoform quantification from rna-seq reads using lightweight algorithms," *Nature biotechnology*, vol. 32, no. 5, p. 462, 2014. 2, 6, 28

[71] R. Petryszak, M. Keays, Y. A. Tang, N. A. Fonseca, E. Barrera, T. Burdett, A. Füllgrabe, A. M.-P. Fuentes, S. Jupp, S. Koskinen *et al.*, "Expression atlas updatean integrated database of gene and protein expression in humans, animals and plants," *Nucleic acids research*, vol. 44, no. D1, pp. D746–D752, 2015. 39

[72] O. Philipp, H. D. Osiewacz, and I. Koch, "Path2ppi: an r package to predict protein–protein interaction networks for a set of proteins," *Bioinformatics*, vol. 32, no. 9, pp. 1427–1429, 2016. 8

[73] A. R. Quinlan and I. M. Hall, "Bedtools: a flexible suite of utilities for comparing genomic features," *Bioinformatics*, vol. 26, no. 6, pp. 841–842, 2010. 14

[74] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of adam and beyond,” *arXiv preprint arXiv:1904.09237*, 2019. 62

[75] G. Rizk, D. Lavenier, and R. Chikhi, “Dsk: k-mer counting with very low memory usage,” *Bioinformatics*, vol. 29, no. 5, pp. 652–653, 2013. 7, 81

[76] A. Roberts and L. Pachter, “Streaming fragment assignment for real-time analysis of sequencing experiments,” *Nature methods*, vol. 10, no. 1, p. 71, 2013. 5

[77] T. Rocktäschel, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, and P. Blunsom, “Reasoning about entailment with neural attention,” *arXiv preprint arXiv:1509.06664*, 2015. 36

[78] M. Rosvall and C. T. Bergstrom, “Maps of random walks on complex networks reveal community structure,” *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008. 13

[79] M. Rosvall, D. Axelsson, and C. T. Bergstrom, “The map equation,” *The European Physical Journal Special Topics*, vol. 178, no. 1, pp. 13–23, 2009. 13

[80] K. Sadakane, “Compressed suffix trees with full functionality,” *Theory of Computing Systems*, vol. 41, no. 4, pp. 589–607, 2007. 74

[81] L. Salwinski, C. S. Miller, A. J. Smith, F. K. Pettit, J. U. Bowie, and D. Eisenberg, “The database of interacting proteins: 2004 update,” *Nucleic acids research*, vol. 32, no. suppl_1, pp. D449–D451, 2004. 40

[82] S. L. Salzberg, M. Pertea, J. A. Fahrner, and N. Sobreira, “Diamund: Direct comparison of genomes to detect mutations,” *Human mutation*, vol. 35, no. 3, pp. 283–288, 2014. 7

[83] M. Sammeth, “Complete alternative splicing events are bubbles in splicing graphs,” *Journal of Computational Biology*, vol. 16, no. 8, pp. 1117–1140, 2009. 19

[84] J. Shen, J. Zhang, X. Luo, W. Zhu, K. Yu, K. Chen, Y. Li, and H. Jiang, "Predicting protein–protein interactions based only on sequences information," *Proceedings of the National Academy of Sciences*, vol. 104, no. 11, pp. 4337–4341, 2007. 35, 62, 63

[85] Y. Silberberg, M. Kupiec, and R. Sharan, "A method for predicting protein-protein interaction types," *PLoS One*, vol. 9, no. 3, p. e90904, 2014. 8, 65

[86] Y. S. Srinivasulu, J.-R. Wang, K.-T. Hsu, M.-J. Tsai, P. Charoenkwan, W.-L. Huang, H.-L. Huang, and S.-Y. Ho, "Characterizing informative sequence descriptors and predicting binding affinities of heterodimeric protein complexes," *BMC bioinformatics*, vol. 16, no. 18, p. S14, 2015. 8

[87] T. Sun, B. Zhou, L. Lai, and J. Pei, "Sequence-based prediction of protein protein interaction using a deep-learning algorithm," *BMC bioinformatics*, vol. 18, no. 1, p. 277, 2017. 8, 36, 62, 63

[88] D. Szklarczyk, J. H. Morris, H. Cook, M. Kuhn, S. Wyder, M. Simonovic, A. Santos, N. T. Doncheva, A. Roth, P. Bork *et al.*, "The string database in 2017: quality-controlled protein–protein association networks, made broadly accessible," *Nucleic acids research*, p. gkw937, 2016. 40

[89] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *arXiv preprint arXiv:1503.00075*, 2015. 36

[90] P. Tonner, V. Srinivasasainagendra, S. Zhang, and D. Zhi, "Detecting transcription of ribosomal protein pseudogenes in diverse human tissues from rna-seq data," *BMC genomics*, vol. 13, no. 1, p. 412, 2012. 6

[91] C. Trapnell, B. A. Williams, G. Pertea, A. Mortazavi, G. Kwan, M. J. Van Baren, S. L. Salzberg, B. J. Wold, and L. Pachter, "Transcript assembly and quantification by rna-seq reveals unannotated transcripts and isoform switching during cell differentiation," *Nature biotechnology*, vol. 28, no. 5, p. 511, 2010. 5, 42

[92] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995. 74

[93] N. Välimäki, W. Gerlach, K. Dixit, and V. Mäkinen, "Compressed suffix treea basis for genome-scale sequence analysis," *Bioinformatics*, vol. 23, no. 5, pp. 629–630, 2007. 74

[94] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008. 73

[95] K. Wang, D. Singh, Z. Zeng, S. J. Coleman, Y. Huang, G. L. Savich, X. He, P. Mieczkowski, S. A. Grimm, C. M. Perou *et al.*, "Mapsplice: accurate mapping of rna-seq reads for splice junction discovery," *Nucleic acids research*, vol. 38, no. 18, pp. e178–e178, 2010. 5

[96] Y.-B. Wang, Z.-H. You, X. Li, T.-H. Jiang, X. Chen, X. Zhou, and L. Wang, "Predicting protein–protein interactions from protein sequences by a stacked sparse autoencoder deep neural network," *Molecular BioSystems*, vol. 13, no. 7, pp. 1336–1344, 2017. 8

[97] B. L. Welch, "The generalization ofstudent's' problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947. 63

[98] B. T. Wilhelm and J.-R. Landry, "Rna-seq-quantitative measurement of expression through massively parallel rna-sequencing," *Methods*, vol. 48, no. 3, pp. 249–257, 2009. 2

[99] L. Wong, Z.-H. You, S. Li, Y.-A. Huang, and G. Liu, "Detection of protein-protein interactions from amino acid sequences using a rotation forest model with a novel pr-lpq descriptor," in *International Conference on Intelligent Computing.* Springer, 2015, pp. 713–720. 8, 40, 62

[100] D. E. Wood and S. L. Salzberg, “Kraken: ultrafast metagenomic sequence classification using exact alignments,” *Genome biology*, vol. 15, no. 3, p. R46, 2014. 7

[101] L. Yang, J.-F. Xia, and J. Gui, “Prediction of protein-protein interactions from protein sequence using local descriptors,” *Protein and Peptide Letters*, vol. 17, no. 9, pp. 1085–1090, 2010. 8, 62, 66

[102] A. Yates, W. Akanni, M. R. Amode, D. Barrell, K. Billis, D. Carvalho-Silva, C. Cummins, P. Clapham, S. Fitzgerald, L. Gil *et al.*, “Ensembl 2016,” *Nucleic acids research*, vol. 44, no. D1, pp. D710–D716, 2015. 37

[103] W. Yin and H. Schütze, “Convolutional neural network for paraphrase identification,” in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2015, pp. 901–911. 36, 62, 63

[104] W. Yin, H. Schütze, B. Xiang, and B. Zhou, “Abcnn: Attention-based convolutional neural network for modeling sentence pairs,” *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 259–272, 2016. 62

[105] Z.-H. You, Y.-K. Lei, L. Zhu, J. Xia, and B. Wang, “Prediction of protein-protein interactions from amino acid sequences with ensemble extreme learning machines and principal component analysis,” in *BMC bioinformatics*, vol. 14, no. 8. BioMed Central, 2013, p. S10. 8, 40, 62

[106] Z.-H. You, L. Zhu, C.-H. Zheng, H.-J. Yu, S.-P. Deng, and Z. Ji, “Prediction of protein-protein interactions from amino acid sequences using a novel multi-scale continuous and discontinuous feature set,” in *BMC bioinformatics*, vol. 15, no. 15. BioMed Central, 2014, p. S9. 8, 40, 62, 63

[107] K. Yugandhar and M. M. Gromiha, “Protein–protein binding affinity prediction from amino acid sequence,” *Bioinformatics*, vol. 30, no. 24, pp. 3583–3589, 2014. 8

[108] Q. Zhang, J. Pell, R. Canino-Koning, A. C. Howe, and C. T. Brown, “These are not the k-mers you are looking for: efficient online k-mer counting using a probabilistic data structure,” *PloS one*, vol. 9, no. 7, p. e101271, 2014. 7, 82

[109] Y. Zhang, W. Chan, and N. Jaitly, “Very deep convolutional networks for end-to-end speech recognition,” in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 4845–4849. 34

[110] Z. Zhang and W. Wang, “Rna-skim: a rapid method for rna-seq quantification at transcript level,” *Bioinformatics*, vol. 30, no. 12, pp. i283–i292, 2014. 6, 16, 28, 38, 49

[111] Z. Zhang, S. Huang, J. Wang, X. Zhang, F. Pardo Manuel de Villena, L. McMillan, and W. Wang, “Genescissors: a comprehensive approach to detecting and correcting spurious transcriptome inference owing to rna-seq reads misalignment,” *Bioinformatics*, vol. 29, no. 13, pp. i291–i299, 2013. 5

[112] T. Zhou, M. Chen, J. Yu, and D. Terzopoulos, “Attention-based natural language person retrieval,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 27–34. 62

[113] H. Zhu, F. S. Domingues, I. Sommer, and T. Lengauer, “Noxclass: prediction of protein-protein interaction types,” *BMC bioinformatics*, vol. 7, no. 1, p. 27, 2006. 8, 65

[114] M. Zou, E. Y. Baitei, A. S. Alzahrani, F. Al-Mohanna, N. R. Farid, B. Meyer, and Y. Shi, “Oncogenic activation of map kinase by braf pseudogene in thyroid tumors,” *Neoplasia*, vol. 11, no. 1, pp. 57–65, 2009. 6