# IMPROVING GENOME ANNOTATION WITH RNA-SEQ DATA

by

Li Song

A dissertation submitted to Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy

Baltimore, Maryland

September, 2018

# Abstract

With the advent of next generation sequencing, researchers can now investigate genome of species and individuals in unprecedented detail. Each part of genome has its own function. Annotation is the process to identify the parts and their functions.

Deep RNA sequencing (RNA-seq) emerged as a revolutionary technology for transcriptome analysis, now widely used to annotate genes. Our transcript assemblers, CLASS and CLASS2, were designed to better detect alternative splicing events and to find new transcripts from RNA-seq data. With sequencing costs dropping, experiments now routinely include multiple RNA-seq samples, to improve the power of statistical analyses. We took advantage of the power of multiple samples in the software PsiCLASS. PsiCLASS simultaneously assembles multiple RNA-seq samples, which significantly improves performance over the traditional 'assemble-and-merge' model.

For many alignment and assembly applications, sequencing errors can confound downstream analyses. We implemented two k-mer-based error correctors, Lighter and Rcorrector, for whole genome sequencing data and for RNA-seq data, respectively. Lighter was the first k-mer-based error corrector without counting and is much faster and more memory-efficient than other error correctors while having comparable accuracy. Rcorrector searches for a path in the De Bruijn graph that is closest to the current read, using local k-mer thresholds to determine trusted k-mers. Rcorrector measurably improves *de novo* assembled transcripts, which is critical in annotating species without a high-quality reference genome.

A newly assembled genome is typically highly fragmented, which makes it difficult to annotate. Contiguity information from paired-end RNA-seq reads can be used to connect multiple disparate pieces of the gene. We implemented this principle in Rascaf, a tool for assembly scaffolding with RNA-seq read alignments. Rascaf is highly practical, and has improved sensitivity and precision compared to traditional approaches using *de novo* assembled transcripts.

Overall, the collection of algorithms, methods and tools represent a powerful and valuable resource that can be readily and effectively used in any genome sequencing and annotation project and for a vast array of transcriptomic analyses.

Thesis committee members:

Dr. Liliana Florea, Johns Hopkins University School of Medicine

Dr. Ben Langmead, Johns Hopkins University

Dr. Sarven Sabunciyan, Johns Hopkins University School of Medicine

# Acknowledgements

First and foremost, I would like to thank my advisor Dr. Liliana Florea. She introduced me to the area of computational biology and helps me on all my research projects. She gave me great freedom to try different methods and made all kinds of mistakes during research, which were valuable lessons for me.

I also would like to thank my advisor Steven Seidel (1951-2014) in Michigan Technological University. From him, I learned how to think like a scientist and this has benefitted me after I left Michigan Tech.

My researches received help from many professors at Hopkins as well. Dr. Ben Langmead kindly guide me through one of my proudest projects, Lighter. Dr. Sarven Sabunciyan helped us in the lasting effort to develop transcriptome assembly methods. I learned many algorithms and mathematical skills from Dr. Vladimir Braverman, Dr. Jim Fill and Dr. Daniel Naiman. I also learned a lot from the experience of participating in a big collaborated project mentored by Jason Miller in J. Craig Venter Institute.

I would love to thank my parents, Jiangqiao Song and Yan Yang. They gave me the best teachings. And without their support, I could not have focused on my research.

Here, I want to thank my friends. Dr. Daehwan Kim is not only one of my best friends, but has also helped me a lot in my research. The experience of collaborating on the project Centrifuge is a treasure to me. Dr. Weilue He was my high-school classmate and roommate in Michigan

Tech. I consulted him many times when I met difficulties in life. Qingyu Xu and Congyuan Yang lead me to have a more colorful life in Hopkins.

Lastly, I need to thank my spirit advisor, Motonari Mouri. His wisdom and resoluteness in difficulty situations inspire me to keep working.

# Table of Contents

# List of Tables

# List of Figures

# Introduction

DNA is the blue print for life. It encodes the proteins that represent the building blocks for an organism. DNA does not generate the proteins directly but through an intermediate, messenger RNA (mRNA). mRNA molecules are transcribed from the DNA, and many of the mRNAs will then be translated into proteins. We call the transcribed portion of the DNA *a gene*. In eukaryotes, such as human, the sequence for the matured mRNA is not contiguous on the DNA, with some portions of the gene being spliced out from the transcribed pre-mRNA sequence. The removed parts are called 'introns' and the remaining parts are 'exons'. Furthermore, due to alternative splicing, pre-mRNAs can splice out different combinations of introns. As a result, a gene has the potential to generate different RNAs, which we will refer as 'transcripts', 'splice variants' or 'isoforms'. Alternative splicing occurs in more than 90% of the genes in human and at similar levels in other eukaryotes. This mechanism makes it possible to produce millions of different proteins from only about 25,000 genes. The process of inferring the function for each piece of the DNA is called genome annotation. In this thesis, we introduce tools that improve on the genome annotation process.

RNA-seq is the next generation sequencing of RNA transcripts. Despite advances in sequencing, a transcript is still much longer than the read length, which is typically 50-150 bp. Therefore, the huge number of short reads need to be pieced together to form the full-length transcripts. Downstream analysis can then be applied to determine their function, expression levels and to compare the sets of transcripts, collectively called the transcriptome, between different

conditions. Reconstructing the full-length transcripts, or 'transcriptome assembly', is a difficult computational problem owing to the short read lengths, biases in the RNA-seq data, and the complexity of splicing. For example, due to alternative splicing two identical reads from the same gene can originate from different transcripts.

There are two general strategies for transcriptome assembly. *Reference-base*d transcriptome assembly uses the alignments of RNA-seq reads on the reference genome. *De novo* transcriptome assembly directly stitching reads into transcripts, guided by their sequence similarity. For a low-expression transcript, there is little overlap between reads and thus such a transcript is more difficult to reconstruct *de novo*. Both strategies are active areas of research.

This thesis will focus on the problem of reference-based transcriptome assembly. We present the CLASS series of reference-based transcriptome assemblers. CLASS is based on the SET_COVER model, which seeks a minimal number of sets (transcripts) to explain all the elements (read alignments). CLASS uses a compact graph data structure to represent a gene and its splice variants, and proposes an efficient dynamic programming algorithm to select a set of likely transcripts from the splice graph or the subexon graph.

As sequencing costs decrease, using multiple RNA-seq samples becomes routine for many biological studies. Multiple RNA-seq samples give more statistical power, for instance for differential expression and differential splicing analyses. Such differential studies use a global annotation (meta-assembly), comprising the expressed genes and transcripts as inferred from the RNA-seq reads. To produce the meta-assembly, the current paradigm is to reconstruct the transcripts for each sample individually, using a single-sample transcriptome assembler, and

then merge the resulting transcript sets across all samples. However, in this approach, each sample is assembled individually, and the merged result conflates the errors, often leading to low accuracy. PsiCLASS proposes an alternative model, in which samples are analyzed simultaneously and gene information is used across the samples to improve both completeness (sensitivity) and reliability (precision). PsiCLASS builds a global subexon graph by merging the sample-wise subexon graphs, filtered to remove likely exon and intron artifacts, and selects a set of transcripts for each sample based on the global splice graph. A unified set of meta-annotations is then obtained from the full set of transcripts by voting. Since the transcripts for each sample are selected from the same underlying global data structure, they are more consistent among samples, as well as with the resulting meta-assembly.

Sequencing errors can affect the performance of tools operating on the raw sequencing data, including alignment tools and the genome assemblers based on De Bruijn graph. This limitation has motivated the development of sequencing error correctors. The most widely used approach for error correction of next generation sequencing data is by leveraging the multiplicity of k-mers. A k-mer is a substring of size k; a read of length r will then contain r-k+1 overlapping k-mers. If a k-mer is error-free ('solid'), it will appear many times in the data set and will have high multiplicity. If a k-mer contains a sequencing error ('weak'), given that the errors are rare and random, it will only occur a small number of times and will have low multiplicity. The error correction consists of turning the weak k-mers into solid k-mers by changing the bases of the read. The tradition approach counts the multiplicity of the k-mers, a process that is slow and memory-consuming. Our tool Lighter is an error corrector based on k-mers without counting. We designed a two-pass streaming algorithm in Lighter to obtain the solid k-mers. The only

sizable data structures in Lighter are two Bloom filters. A Bloom filter is a memory-efficient data structure that can answer the question of whether an element is in a set or not with a small false positive rate. As a result, Lighter is both much faster and much more space-efficient than other error correctors. Furthermore, Lighter consumes constant memory when the sequencing depth for a species increases.

Unlike with whole genome sequencing (DNA) data, where read coverage is expected to be uniform, due to different expression levels of the transcripts in an RNA-seq sample one cannot distinguish solid from weak k-mers with a global multiplicity threshold. For example, a k-mer with low multiplicity can originate from a transcript with low expression level, or can contain an error. Rcorrector adopts a local threshold to distinguish between solid and weak k-mers at k-mer level, and also at read level. To correct a read, Rcorrector searches paths in the De Bruijn graph of read sequences. It then chooses the path whose corresponding sequence has the minimum edit distance to the original read, such that each k-mer's multiplicity in the path exceeds its own local threshold. Thorough evaluation of Rcorrector showed that correcting the errors directly improves the quality of the *de novo* assembled transcripts.

For genomes assembled from short next generation sequencing reads, in particular Illumina reads, the newly assembled sequence is usually highly fragmented into contigs and/or scaffolds. A contig is a portion of the genome that the assembler reconstructed with no gaps. A contig typically terminates at a repeat boundary, when the read or k-mer is not able to resolve the repeat. A mate pair is produced by sequencing the two ends of the same DNA fragment. The two reads are then close on the genome and have well defined position and orientation relative to one another. The difference between the mates' locations on the fragment and on

the (original) genome is called insert size. If the insert size is longer than the factor terminating the contig, such as the repeat size, the mate pair that spans two contigs can be used to connect the two contigs together, with a gap between them. This structure is called a 'scaffold'. A scaffold can contain multiple contigs with specified order and orientation.

Because they contain introns, eukatyotic genes can span thousands of bases and are very likely to be split across different scaffolds. This makes the downstream annotation process difficult. RNA-seq reads are sampled from the gene regions and may 'jump' over the introns, regardless of the insert size, and therefore provide the contiguity information that can be used to connect different pieces of the gene and their underlying contigs and/or scaffolds. While genome sequencing projects have occasionally used *de novo* assembled transcripts to lay out the scaffolds, this process is time-consuming and error-prone, as misassemblies in the transcripts can transfer onto the resulting scaffold. State-of-the-art short RNA-seq read aligners, such as HISAT [1] and STAR [2], can align the short reads on the raw assembly fast and efficiently. Therefore, we developed Rascaf to directly use the alignments of short mate-pair RNA-seq reads to guide scaffolding. Compared with L_RNA_Scaffolder, based on the *de novo* assembled transcripts, Rascaf showed higher sensitivity and significantly better precision.

The rest of the thesis is organized as follows: Chapter 1 presents the reference-based transcriptome assemblers CLASS and CLASS2. Chapter 2 describes PsiCLASS, which extends CLASS2 to simultaneously handle multiple RNA-seq samples. Chapter 3 and Chapter 4 introduce the error correctors Lighter and Rcorrector, for whole genome sequencing and for RNA-seq data, respectively. Lastly, Chapter 5 presents Rascaf, a scaffolder based on RNA-seq alignments.

# Chapter 1

# CLASS: accurate and efficient splice variant annotation from RNA-seq reads

## 1.1 Introduction

Alternative splicing is an inherent property of eukaryotic genes, with important roles in increasing functional diversity and in disease [3, 4, 5]. More than 90% of the human genes are alternatively spliced [6, 7], with similar levels reported in other eukaryotes. Each gene can produce from one to potentially thousands of splice variants under different cellular conditions, and gene splice isoforms can have similar, independent and even antagonistic functions. Identifying the genes and their transcript variants is therefore a critical first step in answering a broad range of biological questions. Over the past years, next generation sequencing of cellular RNA (RNA-seq) has enabled the discovery of thousands of novel non-coding RNAs and has significantly expanded our catalog of splice variants. However, despite significant progress, extracting gene expression estimates and identifying splice variants in the vast amounts of short read data remains challenging, demanding bioinformatics tools that are fast, accurate and efficient.

The primary goal of a typical RNA-seq analysis is to comprehensively determine the precise exon–intron boundaries on the genome for all transcripts and to estimate their expression

levels in the samples. Before this can be accomplished, reads must be mapped to the genome with a fast spliced alignment program that accounts for introns and sequencing errors [8]. Alignments are then pieced together to form gene and transcript models. Virtually all genome-guided transcript assemblers build a graph that represents a gene and its splice variants, and then traverse it to select a subset of transcripts that are likely represented in the sample. Among current programs, Cufflinks [9] connects overlapping reads into overlap graphs, Scripture [10] and IsoLasso [11] build connectivity graphs, and iReckon [12], Scripture and SLIDE [13] generate splice or subexon graphs [14]. Although there are some differences among the exons and introns predicted by each program, these representations more or less encode equivalent sets of candidate transcripts. Therefore, the strategy for selecting transcripts from among the many encoded possibilities in the graph is important for the program's accuracy as well as for the number of variants identified. Parsimony-based methods such as Cufflinks' minimum partition algorithm select a mathematically minimum number of transcripts. They can usually identify the genes and most major isoforms relatively accurately, but are less apt at identifying low abundance splicing events. 'Best fit' methods, which include IsoLasso, SLIDE and iReckon, choose a subset of transcripts such as to optimize an objective function, using either an integer programming or an expectation maximization formulation. The main problem with these approaches is over-fitting, where programs tend to report a large number of spurious transcripts based on low abundance reads. In yet another category, programs such as SpliceGrapher [15] simply omit enumerating transcripts altogether, or otherwise exhaustively enumerate all splice variants encoded in the graph (Scripture). While they can generally capture a larger portion of the true splicing variation, these methods are too imprecise to allow

meaningful downstream analyses. Lastly, programs differ in their use of known annotations to inform their predictions. Annotation-guided methods, such as iReckon and SLIDE, rely on an existing set of gene annotations to build their gene models. For species for which there is already an extensive set of gene annotations these methods generally produce more variants, but are also more prone to reporting spurious isoforms and cannot be used to identify novel genes. In contrast, *de novo* programs including Cufflinks, Scripture and IsoCEM, build gene and transcript models from RNA-seq reads alone, without any prior knowledge of gene structure, and therefore are more suited to annotate newly sequenced or less studied organisms. Overall, while many tools already exist to determine the expressed genes and loci in an RNA-seq sample, there is an unmet need for methods that specifically target alternative splicing.

We developed CLASS and its successor CLASS2 (Constraint-based Local Assembly and Selection of Splice variants), to bridge this gap and detect low abundance splice variation with high accuracy. At its core is the concept of the splice graph, a data structure that we have previously employed in splice variant annotation using both conventional Sanger (EST) [16] and next generation sequencing [17]. A splice graph compactly represents a gene with its exons as nodes and introns as edges; splice variants can be read as maximal paths in the graph. CLASS2 uses a linear programming method to predict exons, and then connects them into splice graphs via introns detected from spliced alignments. Since the splice graph may encode many biologically unfeasible combinations, CLASS2 uses an efficient dynamic programming optimization algorithm to select candidate transcripts. CLASS2 builds upon its predecessor CLASS [17], but brings several critical algorithmic and performance improvements, including a new formulation for transcript scoring and selection as an optimization problem, novel and scalable dynamic

programming transcript selection algorithms, and a new model for 'intronic' noise due to reads from unspliced RNA. This chapter will focus on CLASS2. When compared to reference programs, CLASS2 captured significantly more splicing variation, both fully reconstructed transcripts and partial splicing events, with high precision. Most importantly, it was the only program tested that produced consistently well formed and easy to interpret annotations for all applications and sequencing strategies. More specifically, our comparative analyses have shown that:

1. CLASS2 offers the best tradeoff between sensitivity and precision in reconstructing full transcripts. In its default setting, CLASS2 detects 10–70% more transcripts than Cufflinks, which is the most popular and most precise of these programs, with higher or comparable precision. In its sensitive settings, CLASS2 detects up to twice as many transcripts as Cufflinks for a relatively small drop in precision.
2. It is the best suited to capture local alternative splicing variation. In particular, it can detect up to twice as many alternative splicing events as Cufflinks, with high precision. CLASS2 finds slightly fewer events than Scripture, which is the most sensitive of the programs, but its precision is considerably (70–80%) higher.
3. It employs a combined gene-level and genome-level model of intronic 'noise' that allows more accurate detection of intron retention events.
4. The amount of novel alternative splicing variation detected by CLASS2 increases with increasingly large data sets.
5. CLASS2 is multi-threaded and scales well with the amount of data, requiring <3GB RAM for all of our tests, and can complete most regular tasks in a few hours.

6. Lastly, since CLASS2 can produce annotations from RNA-seq data alone, without requiring an existing set of gene annotations, it is very well suited for the annotation of newly sequenced organisms.

We present the overall strategy below, followed by more details about the individual algorithms in the corresponding Methods sections. We then comparatively evaluate CLASS2 and several popular programs, including both *de novo* and annotation-dependent transcript assemblers, on both control and real RNA-seq sets, in the Results section.

## 1.2 Methods

### 1.2.1 Overview

CLASS2 determines a set of transcripts in three stages (Figure 1-1). First, it infers a set of exons from read coverage levels and splice sites using a linear programming technique. Then, it connects the exons into a splice graph via introns extracted from spliced reads. Once the graph is constructed, CLASS2 selects a subset of transcripts from among those encoded in the graph using an efficient splice graph-based dynamic programming algorithm.

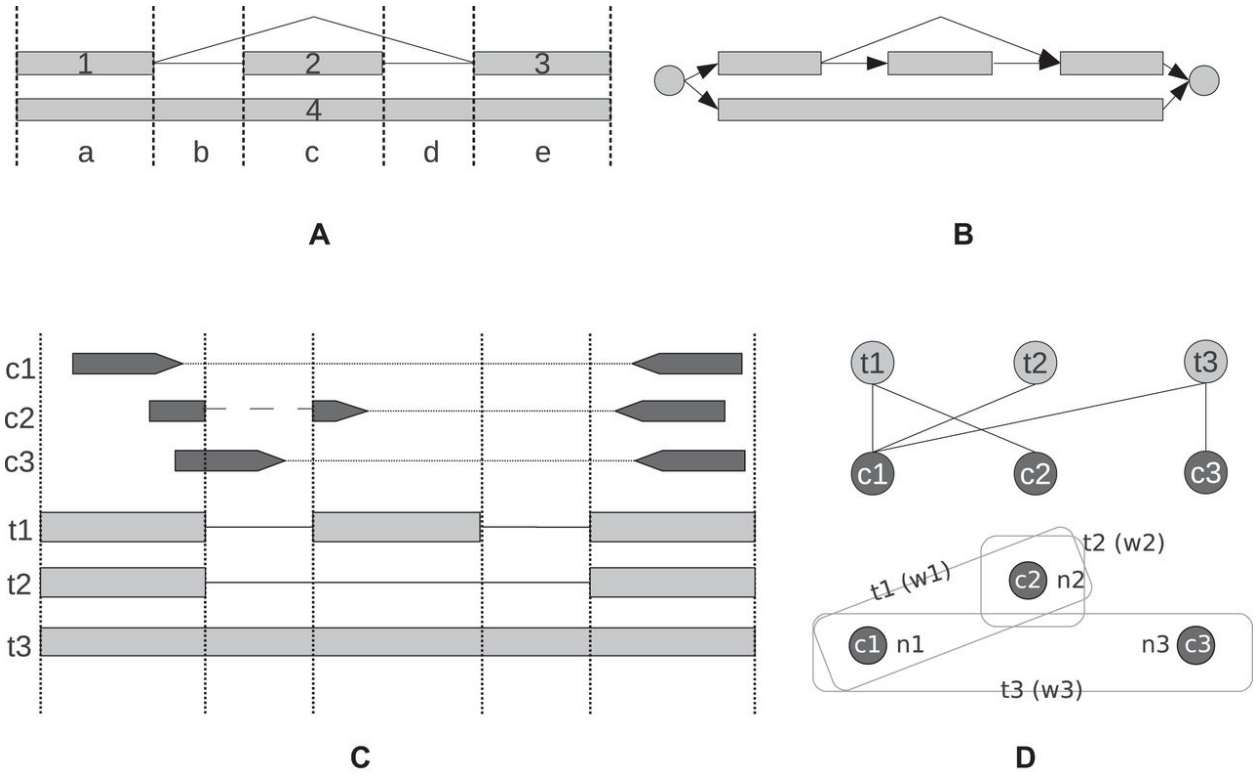



*Figure 1-1 The CLASS2 transcript assembly algorithm*

*(Step 1 (A) Exon and introns. Infer exons from the read coverage levels, using linear programming, and introns from spliced alignments. Step 2 (B) Splice graph. Build a splice graph to represent the gene, connecting exons by introns. Shown is a section from a splice graph, with a skipped exon event and a 2-intron retention event, encoding two possible paths (transcripts). Step 3 (C) Constraints. Cluster reads into classes (constraints) by their splicing and interval patterns. Step 4 (D) Transcript selection. Build and solve the bipartite constraint graph and associated transcript selection problem, shown here for three read pairs c1, c2 and c3, and three transcripts t1, t2 and t3.)*

### 1.2.2 Building the exons

Exons are key to the transcript assembly process, because incorrectly reconstructing exons can miss important gene variations or can create false ones. Since current RNA-seq reads are too short to cover many exons end-to-end, CLASS2 uses read coverage levels along the genome and splice junctions from spliced read alignments to find exons (Figure 1-1A). CLASS2 employs a two-step procedure to determine a set of exons: first, it enumerates all combinations of exons

that can explain the splice site patterns and paired-end reads. Second, for each such combination it formulates and solves a linear program expressing several types of constraints. Intuitively, the read coverage levels for all alternative exons over a common interval should cumulatively add up to the observed read coverage levels. Additionally, we assume read coverage levels are locally uniform, and therefore the coverage of adjacent portions of the same exon should be similar. Each exon combination is scored by the linear program, and the combination with the minimum objective function value is chosen in the end.

### 1.2.3 Modeling intronic 'noise'

Intronic RNA, produced by unspliced pre-mRNA transcripts that are either residual or part of the experiment, is a common artifact with real RNA-seq samples. Such intronic 'noise' can confound the detection of mature mRNA resulting from intron retention and alternative transcription start and termination events [18]. Distinguishing between 'signal' and 'noise' is therefore critical for creating a full and accurate set of exons. CLASS2 introduces a new method to identify intronic mRNA, by modeling intronic read levels across genomic intervals, both within a gene locus and along the genome. The gene-level 'noise' is modeled as a Poisson distribution of the individual intronic positions, retaining the introns whose average coverage ranks at the top of the distribution (P-value = 10e−5). For the genome-wide 'noise', we consider the coverage distribution of intronless regions ('islands'), modeled as a normal distribution, and retain only those introns with Z-score > 6. In the end, only introns that pass both filters are deemed as likely retained introns or alternative gene ends, and are then incorporated into the exon finding procedure.

### 1.2.4 Transcript enumeration and selection

Once a set of exons is determined, CLASS2 generates a splice graph by connecting the exons (nodes) via introns (edges) extracted from spliced read alignments. Candidate transcripts are encoded in the graph as maximal paths from a node with no incoming edges (source) to a node with no outgoing edges (sink) (Figure 1-1B). Since the splice graph generally encodes a much larger number of transcripts than is biologically possible, CLASS2 uses a selection procedure to identify a subset of candidates that can explain all contiguity constraints from spliced reads and paired-reads. In practical terms, a constraint is a cluster of reads or read pairs that share the same set of exons or exon fragments and therefore can be assembled into the same transcript (Figure 1-1C).

Conceptually, we model the problem as a graph with two types of nodes (bipartite), transcripts and constraints, where each transcript node is connected by edges to the constraints it satisfies, and we must select a subset of transcripts that collectively satisfy all constraints (Figure 1-1D). In early work, we implemented a simple greedy SET_COVER approximation algorithm [17] that aimed to minimize the number of transcripts that could explain all the read patterns, or constraints, without regard to the number of supporting reads. Here, we report an improved algorithm that additionally takes into account the read coverage (abundance) information for each transcript and constraint, modeled as a dynamic programming optimization problem. It selects a subset of candidate transcripts while simultaneously assigning a set of compatible reads, and it does so efficiently by exploiting the compactness of the splice graph data structure. The algorithm iteratively grows a set of transcripts by selecting, at each step, the transcript that maximizes a scoring function which takes into account both the

number of constraints not covered by the current set and their abundance. As a new transcript is selected, reads are simultaneously assigned to it as determined by its set of constraints, and the algorithm is reiterated with the updated sets of constraints and transcripts.

Since the algorithm favors abundant isoforms, transcripts are being selected largely in the order of their abundance, from the most highly expressed to the least expressed. This allows the selection procedure to be terminated whenever the abundance reaches a user-specified cutoff (parameter '-F'), with the most trusted isoforms being reported first. To further improve the algorithm's efficiency for genes with complex structures, rather than enumerating all transcripts at each step in the algorithm, CLASS2 implements an efficient splice-graph based dynamic programming transcript selection procedure, described below. This method considerably reduces both memory and run time, and allows the program to be run on very large data sets without sacrificing sensitivity.

### 1.2.5 Exon reconstruction algorithm

To determine a set of exons, CLASS2 uses a protocol similar to CLASS (15), modified to exclude intervals that contain intronic 'noise'. More specifically, it analyzes regions of the genome covered by reads, which represent exons or combinations of exons, using splice sites to split each region into intervals. Each interval can belong to more than one exon; the portion of an exon corresponding to an interval is called subexon. To determine the most likely combination of exons within a region, CLASS2 enumerates all feasible exon sets, i.e. that are necessary and sufficient to explain all splice sites and all reads. For each such set it formulates and solves a

linear program (LP), which is used to score the combination. The combination with the best LP score is chosen as the representative set of exons. The linear program is formalized below.

### 1.2.6 The LP-based scoring system for exon combinations

More formally, consider a region R = $r_1 \ldots r_N$, where N is the number of intervals. Let $L_j = |r_j|$ be the length of interval $r_j$, and $L = \sum_{j=1}^{N} L_j$ the length of the region. Denote S = {$X_1$, …, $X_M$} the set of possible exons, represented as vectors: $Xi=(x_{i,j}) \in \{0,1\}^N$, with $x_{i,j} = 1$ if and only if $X_i$ contains interval $r_j$, and 0 otherwise. Hence, $X_i$ will contain all 0s except for a run of 1s, starting at interval $b_i$ and ending at interval $e_i$. Given a candidate set of exons S' ⊆ S, CLASS2 assigns each subexon an (unknown) read coverage level, $c_{i,j}$, defined as the average number of reads per base of subexon i,j. Let $C_j$ be the (observed) read coverage on interval j. We write a linear system with the following constraints:

(i) additivity - for each interval $r_j$, j = 1,…,N, the cumulative coverage levels of subexons within the interval should be roughly equal to the observed coverage level:

$$\left| \sum_i x_{i,j} c_{i,j} - C_j \right| \leq \epsilon_j$$

(ii) continuity - for each exon Xi ∈ S', the coverage of adjacent subexons should be roughly equal:

$$\left| c_{i,j} - c_{i,j-1} \right| \leq \epsilon_{i,j} \text{ for each } j = b_i + 1, \ldots, e_i$$

(iii) conservation - the total coverage of all exons should be roughly equal to the total coverage of the region:

$$\left|\sum_i \sum_j c_{i,j} L_j - \sum_j C_j L_j\right| \leq \epsilon$$

(iv) non-negativity - all (sub)exons of exons should be expressed:

$$c_{i,j} \geq 1, if\ x_{i,j} = 1; c_{i,j} = 0\ if\ x_{i,j} = 0$$

The objective function minimizes the total error:

$$min_{i,j}(\sum_j \epsilon_j + \sum_{i,j} \epsilon_{i,j} + \epsilon)$$

For single-end reads, this value is used explicitly to score the combination. For paired-end reads, deviations from the observed fragment length distribution are included as penalties to more finely differentiate among likely exon sets, as described in [17]. In the end, the exon combination with the smallest score ('error') is chosen.

Here is an example illustrating the algorithm. Consider the region in Figure 1-1A, we have:

(i) Additivity: $| c_{1,a}+c_{4,a} - C_a | \leq \varepsilon_a$ , $| c_{4,b} - C_b | \leq \varepsilon_b$, $| c_{2,c}+c_{4,c}-C_c | \leq \varepsilon_c$, $| c_{4,d}-C_d | \leq \varepsilon_d$, $| c_{3,e}+c_{4,e}-C_e | \leq \varepsilon_e$

(ii) Continuity: $| c_{4,a}-c_{4,b} | \leq \varepsilon_{4,a}$, $| c_{4,b}-c_{4,c} | \leq \varepsilon_{4,b}$, $| c_{4,c}-c_{4,d} | \leq \varepsilon_{4,c}$

(iii) Conservation:

$$| (c_{1,a}+c_{4,a}) L_a + c_{4,b} L_b + (c_{2,c}+c_{4,c}) L_c + c_{4,d} L_d + (c_{3,e}+c_{4,e}) L_e - (C_aL_a+C_bL_b+C_cL_c+C_dL_d+C_eL_e) | \leq \varepsilon$$

(iv) Non-negativity: $c_{1,a} \geq 1$, $c_{4,a} \geq 1$, $c_{4,b} \geq 1$, $c_{2,c} \geq 1$, $c_{4,c} \geq 1$, $c_{4,d} \geq 1$, $c_{3,e} \geq 1$, $c_{5,e} \geq 1$

Optimization function: min ( $\Sigma_{j \in \{a-e\}} \varepsilon_j + \Sigma_{i=1,4;\ j \in \{a-e\}} \varepsilon_{i,j} + \varepsilon$ )

Once determined, exons are connected into a splice graph via introns extracted from spliced alignments, and candidate transcripts are enumerated as maximal paths in this graph. The candidate transcript set is typically much larger than the true set of transcripts. CLASS2 implements several algorithms to select a subset of transcripts that are the most likely to be represented in the sample.

### 1.2.7 Transcript selection algorithms

The goal is to select a subset of the transcripts encoded in the splice graph that can collectively explain all the reads, which we formulate as a dynamic programming optimization problem. We implement an iterative procedure that simultaneously selects the next transcript and assigns reads to it, thus estimating its abundance in the process. To start, we mark the boundaries of the exons along the genomes and divide the gene into intervals, as described above. To reduce space, we group reads (or read pairs) that cover the same set of intervals into classes, called constraints. For each constraint $c_i$, we define its abundance $a_i$ as the number of reads (or read pairs) for that constraint divided by the number of possible start positions of the reads within the intervals. Each constraint can be included (satisfied) into one or more candidate transcripts, $c_i \sim t_j$; conversely, a transcript can be viewed as the set of constraints it satisfies: $t_j = \{c_1, \ldots, c_{n1}\}$. We then denote the abundance of a transcript, $A_j$, as the minimum abundance of its set of constraints: $A_j = \min\{a_i \mid c_i \sim t_j\}$ Let G be a graph with n transcripts $T = \{t_1, \ldots, t_n\}$ and m constraints $C = \{c_1, \ldots, c_m\}$. We give a basic enumeration and selection algorithm for relatively simple graphs, and then an efficient splice-graph based implementation that can efficiently handle complex graphs, below.

### 1.2.8 Basic algorithm

For a small graph, it is feasible to enumerate and assess all candidate transcripts $t_1$, …, $t_n$ encoded in the graph. At each step, the algorithm evaluates all remaining transcripts and selects the new transcript, $t_i$, that maximizes the score function $V_i = n_i/(2-A_i /\max A)$, where $n_i$ is the current number of constraints that transcript $t_i$ is compatible with, $A_i$ is the abundance of transcript $t_i$, and max A is the maximum abundance over all transcripts of the gene. Once a transcript $t_i$ is selected, its abundance is subtracted from those of the constraints it satisfies: $c_j = c_j - A_i$. If for any constraint the abundance becomes 0, it is removed from the set. The algorithm is reiterated until there are no non-empty constraints.

### 1.2.9 An efficient splice graph-based algorithm

For complex genes that can generate a large number of transcripts, it may not be efficient or even feasible to enumerate and assess all transcripts at each step. Instead, we take advantage of the compactness of the splice graph representation and the locality of the constraints to design a memory and time efficient dynamic programming algorithm. We start by giving an algorithm to iteratively find the next transcript $t_i$ that satisfies the maximum number of constraints $n_i$, by traversing the graph while calculating an optimal path, and then modify it to take into account the abundance, or read numbers.

Let L be a subpath (subtranscript) in the splice graph and L' the minimum subpath immediately following L such that the constraints partially compatible with L cannot end after L' ('memory'). We enumerate all the paths L' and recursively calculate the maximum number of constraints f(L) of subtranscripts starting with subpath L:

$$f(L)=\max\{ f(L')+c(L,L'), \text{ if } L' \text{ exists};\ c(L), \text{ if } L' \text{ does not exist}\},$$

where c(L,L′) is the number of constraints partially compatible with L (start within L) and compatible with the concatenated subpath L.L′, and c(L) is the number of constraints covered by subtranscript L. The algorithm starts with considering every 5′ exon as a subpath. Along with the maximum number of constraints covered, the algorithm can also track the corresponding optimal transcript. Note that, while iterating over subpaths of the splice graph does not change the theoretical exponential complexity of the algorithm, due to the limited fragment size the number of possible sub-paths is drastically reduced, leading to significant savings in both memory and run time in practice. An example illustrating the procedure is given in

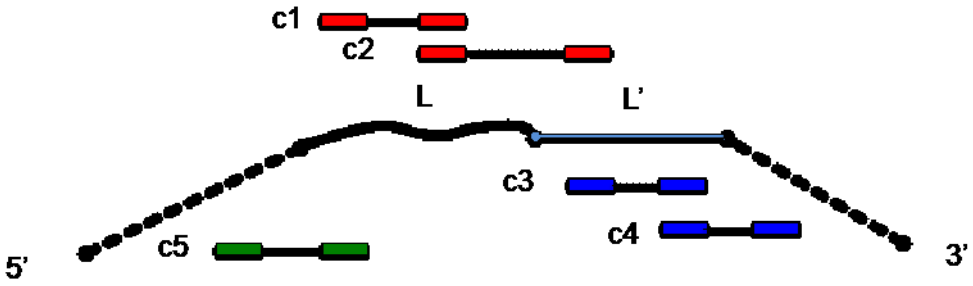



*Figure 1-2 Dynamic programming representation of transcript selection algorithm*

*(Starting from a 5' end, the algorithm calculates for each subpath L the maximum number of constraints for transcripts starting with L and ending at a 3' end: f(L) = max { f(L') + c(L,L') }, where L' is the minimum subpath such that all constraints starting in L cannot end past L' (i.e., L' is used as 'look-ahead'). On the figure, $c_1$ and $c_2$ are constraints compatible with and used in the calculation of f(L). $c_3$, $c_4$ and $c_5$ are not used in this stage, however, $c_3$ and $c_4$ had been used in the calculation of f(L').)*

To incorporate abundance information into the optimization process, we modify the algorithm as follows. When processing L and L', we exclude subpaths that cover constraints with abundance less than or equal to some fixed value x. Hence, the algorithm reports the transcript

covering the most constraints among those whose abundance is larger than x. We call such a transcript an x-abundance transcript. This variation helps determine, at each step, the transcript t* with $\max_i V_i = \max_i n_i /(2-(A_i /\max A))$. We first calculate the 0-abundance transcript; suppose its abundance is x1. We then calculate the $x_1$-abundance transcript, and so on, until we cannot find any $x_m$-abundance transcript, where $x_m = \max A$, in the (m+1)-st iteration. Then the following Theorem establishes that transcript t* is among the transcripts computed.

Theorem: The optimal transcript t* is among the 0-abundance, $x_1$-abundance, …, $x_{m-1}$-abundance transcripts.

Proof: Suppose n*, V*, A* corresponds to the number of covered constraints, score and abundance for the optimal transcript t*. Let x0 = 0, then the following two properties hold from the definitions above: (i) $0 = x_0 < x_1 < \cdots < x_m$; and (ii) $n_0 \geqslant n_1 \geqslant \cdots \geqslant n_m$. Then A* is between $x_0$ and $x_m$, and suppose that $x_i < A^* \leqslant x_{i+1}$, where $0 \leqslant i < m$. Denote the $x_i$-abundance transcript by $t_i$. Then $V_i \leqslant V^*$, by virtue of the fact that t* is the optimal transcript. We only need to prove that $V_i = V^*$.

Suppose $V_i < V^*$, and we already know that $n_i \geqslant n^*$ because the dynamic programming always returns the transcript covering the most constraints (property (2) above). According to the definitions of $V_i$ and V*, then it is necessary that $A_i < A^*$ in order to make $V_i < V^*$. But, $A_i = x_{i+1}$ based on the definition. Therefore, $A^* > Ai = x_{i+1}$, which contradicts the fact that A* is in the interval $(x_i, x_{i+1}]$. Hence, the assumption that $V_i<V^*$ is false and we must have $V_i = V^*$ and hence $t^* = t_i$ (i.e. the $x_{i+1}$-abundance transcript), which concludes the proof.

### 1.2.10 Materials and sequences

For our analyses on simulated data, we generated RNA-seq reads with the software FluxSimulator [19], starting from the GENCODE v.17 gene annotations and choosing the options 'RNA fragmentation' and 200 million clusters. In total, 15 062 genes and 22 544 GENCODE transcripts were represented by the 200 million 75 bp paired-end reads in the sample. Directional mRNA and rRNA-depleted sequencing libraries were prepared from peripheral blood lymphocyte (PBL) samples from the same individual, collected as part of a neuropsychiatric study in twins, and were subjected to paired end sequencing. The sequencing produced 183 million and 317 million 100 bp paired-end reads, respectively. Lastly, for our analyses on very deep sequencing data sets, RNA-seq reads from long RNAs in whole-cell, cytosol and nucleus (two biological replicates each) of IMR90 lung fibroblast cells were downloaded from the ENCODE project's website at UCSC (http://genome.ucsc.edu/ENCODE). All reads were mapped to the human genome hg19 using the software Tophat2 [20] using a combined non-redundant set of GENCODE and RefSeq transcripts as reference annotations and all other default parameters.

### 1.2.11 Analysis of alternative splicing events

To evaluate the programs for their ability to capture individual types of alternative splicing events, we generated a reference set of events (exon skipping, intron retention and alternative exon ends) from the simulated data. We used ASprofile [21] to extract events from the transcripts sampled by FluxSimulator, and then filtered them to retain only those actually supported by the reads in the sample. We processed each program's GTF output in a similar manner and compared against the reference sets. To characterize the sources of errors, we

searched the set of false positive predictions from each program against the set of events extracted from the full GENCODE data set, which determine artifacts due to paralogs and splice variants present in the annotation. The remaining false positive events were searched for spurious introns and for class-specific patterns. For intron retention, these include mis-classification of 5' and 3' terminal exons and of reads from alternative exons overlapping the intron, whereas for alternative exon ends they include spurious chimeric combinations of exon start and ends.

### 1.2.12 Evaluation measures

We used conventional measures, as introduced in [11], to assess program performance at the transcript, exon, intron and alternative splicing event levels: Recall (sensitivity) = TP/(TP+FN), Precision = TP/(TP+FP) and F-value = 2*Recall*Precision/(Recall+Precision).

### 1.2.13 PCR validation of predicted intron retention events

PCR validation of predicted IR events was performed by Dr. Sarven Sabunciyan, Department of Pediatrics, JHU. Human Blood (Clonetech CAT#:636592) and human genomic DNA (Promega Madison, WI CAT#:G1471) were purchased from the suppliers. One microgram of total RNA was converted into cDNA using the SuperScript First-Strand Synthesis kit (Life Technologies CAT#:11904018) following the manufacturers recommended protocol. Q5 High Fidelity DNA polymerase (NEB Cat#:M0491S) was used to generate amplicons in gDNA and cDNA. The primers were ordered from Integrated DNA Technologies (Corlaville, IA, USA); primer sequences are listed in Table 1-1.

*Table 1-1 Primer sequences for PCR validation*

| PCR | Primer Name | Sequence |
|---|---|---|
| CACNA2D4-1 | CACNA2D4_A1For_94 | GGGTCCTGCTTCTTGTGTTT |
| | CACNA2D4_A1Rev_336 | GACAGTGGGGATAGGTGACC |
| CACNA2D4-2 | CACNA2D4_A2For_57 | CACGAACACGGGGTACTCC |
| | CACNA2D4_A2Rev_175 | TGCCACCATGTTTTCCTGTG |
| KLRF1-1 | KLRF1_A1For_165 | GGAGTTCTGCCCAAACATCTC |
| | KLRF1_A1Rev_354 | ACTGTGGAGTGTACTAATAGAGC |
| KLRF1-2 | KLRF1_A2For_172 | TGCCCAAACATCTCAACTTACA |
| | KLRF1_A2Rev_421 | CCGTATTAGACTGTATGCCACT |

Fifty microliters of PCR reactions were performed containing 1x Reaction buffer, 0.5 µM of each primer, 1 unit of Taq and 200 µM of each dNTP. Each PCR reaction also contained approximately one-tenth of the cDNA synthesis reaction or 75 ng of genomic DNA. Using an annealing temperature of 65°C, the PCR reactions were amplified for 35 cycles. The resulting amplicons were cloned into the Topo 2.1 cloning vector (Life Technologies) and individual clones were sequenced at the Johns Hopkins Sequencing and Synthesis Core Facility.

## 1.3 Results

### 1.3.1 Comparative evaluation on control data

We evaluated CLASS2 and several state-of-the-art programs for their ability to reconstruct full transcripts and to capture partial splice variation. We included in our tests four *de novo* assemblers, namely CLASS2 (v. 2.1.2), Cufflinks (v. 2.1.1; [9]), IsoCEM (v. 0.9.1; [11]) and

Scripture (v. beta2; [10]), and two annotation-based methods, SLIDE (May 7, 2012 download; [13]) and iReckon (v. 1.0.7; [12]). We ran CLASS2 in two different modes, stringent (default; '-F 0.05') and sensitive ('-F 0.01'); the latter allows the program to report more minor isoforms. For the annotation-based programs we provided GENCODE v.17 [22] gene annotations as guides. To generate test data, we simulated 200 million 75 bp paired end reads using FluxSimulator [19] and starting from GENCODE v17 gene annotations as models. Reads were then mapped to the human genome hg19 using the program Tophat2 [20] and assembled with each program.

### *Performance of programs in detecting full-length transcripts*

To evaluate the performance and also to identify potential limitations and biases of each program we performed two types of analyses. In the first analysis we compared the transcripts produced by each program against the set of transcripts sampled by FluxSimulator, to obtain an unbiased assessment. We then also compared the predictions against a comprehensive set of non-redundant GENCODE and RefSeq transcript models, to identify biases and artifacts due to annotated paralogs and splice variants representing alternative combinations of the same exons. These classes of artifacts are impossible to tease apart on real data, where the ground truth is not known, and will be erroneously counted as true matches, thus over-estimating the program's performance. When evaluated against the set of true annotations (Figure 1-3A), most programs detect a majority (63–78%) of the exons and introns ('set of parts') of the sampled transcripts, with the notable exception of iReckon, which only finds roughly 52% of the features in each category. SLIDE is the most sensitive among the programs but has very low precision, and Cufflinks and CLASS2 are the most precise. CLASS2 and CLASS2_F0.01 have the best overall performance, detecting a large fraction of both exons and introns with remarkably

high precision, >90% for exons and >97% for introns. Programs rank similarly for reconstructing full-length transcripts. CLASS2 and CLASS2_F0.01 again have the best overall performance as measured by the F-value, a combined measure of sensitivity and precision, and are able to reconstruct 9% and 16% more full-length transcripts compared to Cufflinks, the next and close runner up.

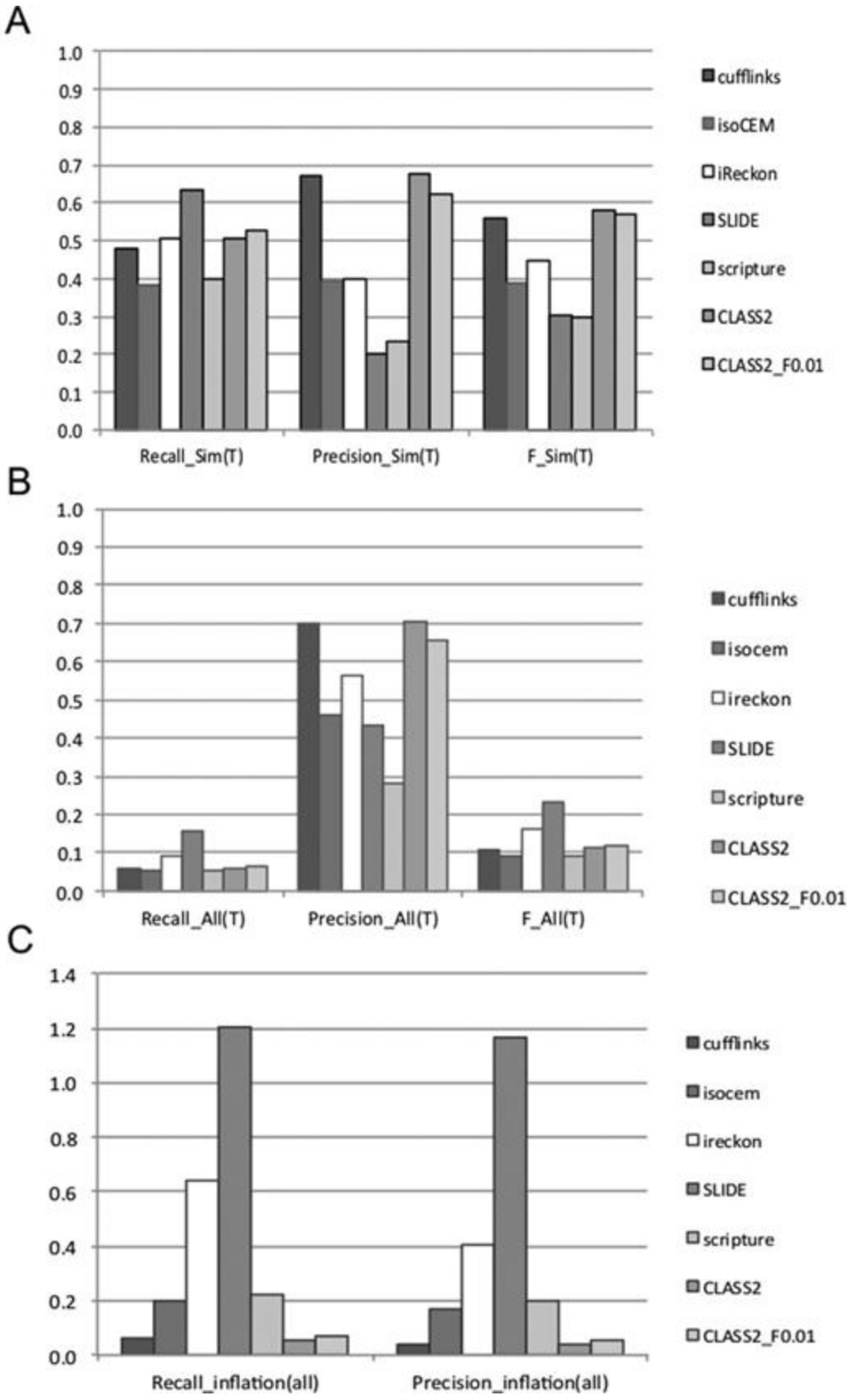

A
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Recall_Sim(T)
Precision_Sim(T)
F_Sim(T)
cufflinks
isoCEM
iReckon
SLIDE
scripture
CLASS2
CLASS2_F0.01
B
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Recall_All(T)
Precision_All(T)
F_All(T)
cufflinks
isocem
ireckon
SLIDE
scripture
CLASS2
CLASS2_F0.01
C
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Recall_inflation(all)
Precision_inflation(all)
cufflinks
isocem
ireckon
SLIDE
scripture
CLASS2
CLASS2_F0.01

*Figure 1-3 Performance of programs in reconstructing full-length transcripts, on simulated data*

*(Observed performance values when measured (A) against the set of FluxSimulator-sampled transcripts ('truth'), and (B) against the full set of GENCODE reference annotations. Recall = TP/(TP+FN), Precision = TP/(TP+FP) and F = 2*Recall*Precision/(Recall+Precision). (C) Performance 'inflation', or the difference between performance measured on the full GENCODE set and the subset of GENCODE transcripts actually represented in the sample. The additional matches are from spurious paralogs and variants not present in the sample. PCI = (Match_GENCODE/Match_sim) -1, where Match_sim refers to the subset of transcripts actually present in the simulated sample.)*

In our second analysis, evaluating the programs against the full set of GENCODE and RefSeq gene annotations revealed several types of biases and errors (Figure 1-3B,C). All programs now seemingly detect the 'parts' equally well (∼20% sensitivity and 88–100% precision), indicating that many of the false predictions in the earlier comparison come from paralogs of the genes in the sample. Unsurprisingly, programs also seemingly detect more of the reference transcripts, artificially increasing programs' performance. In particular, the two annotation-based methods show the largest inflation, with SLIDE more than doubling (120% increase) the number of annotation matches and iReckon adding 64% more matches, by virtue of their use of known annotations to scaffold gene models. When we traced these additional matches, most were variants of the sampled genes (53–92%), and the rest were paralogs, except for iReckon where the variants and paralogs each accounted for roughly half of the false matches. A large portion of the artifacts, between 15% and 67% of the total (with the exception of SLIDE, which had very few), were single exon transcripts. However, even when restricting our analysis to multi-exon transcripts only, SLIDE had very high inflation (128%), followed by iReckon (25%) and Scripture (22%). CLASS2 (both variations) and Cufflinks had the lowest inflation by far, between 5–7%. Thus, these two programs are the most trusted to produce measurable results on real data.

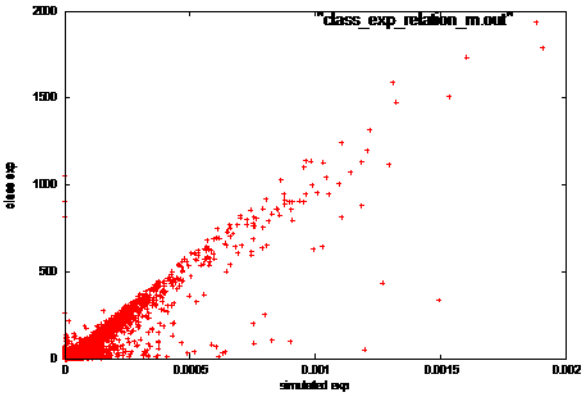
### *Performance of programs by transcript abundance*

We further assessed the performance of programs as a function of the abundance level of transcripts. Simulated transcripts were divided into high-, medium- and low-abundance groups based on their relative expression level assigned by FluxSimulator (FPKM $\leqslant$ 5e−7, low; 5e−7 < FPKM $\leqslant$ 0.0001, medium, FPKM > 0.0001, high). Because the programs do not classify their output into classes, true precision values cannot be computed. However, we calculate a measure of precision based on the full set of predicted transcripts. Programs' performance was more varied across the three ranges, with SLIDE being the most sensitive and CLASS2 a close second for the high and medium abundance transcripts, whereas the precision was 4-fold higher for CLASS2. The two annotation-based programs iReckon and SLIDE were best suited for the low-abundance class (Table 1-2). All programs, especially *de novo* assemblers, had difficulty reconstructing low-abundance transcripts, many of which did not have sufficient reads to cover their entire length. Overall, CLASS2 exhibited the best overall performance for the medium- and high-abundance transcripts, and performance comparable to *de novo* assemblers for the low-coverage transcripts. Lastly, while CLASS2 does not explicitly address the problem of transcript quantification, there is a high correlation between the abundance values of full-length reconstructed transcripts estimated with CLASS2 and the FluxSimulator generated expression levels (R2 = 0.972), surpassed only by IsoCEM (R2 = 0.977) (Figure 1-4 Correlation of predicted and sampled ('truth') expression values for transcripts). Overall, CLASS2 can reconstruct most high and medium expression isoforms of a gene, where it is the best or comparable overall program, as well as some of the rare isoforms.

*Table 1-2 Programs performance by transcript abundance class*

| | | High (2402 transcripts) | | Medium (13 464 transcripts) | | Low (6658 transcripts) | |
|---|---|---|---|---|---|---|---|
| Program | Transcripts | R | P | R | P | R | P |
| CLASS2 | 16 790 | 0.824 | 0.118 | 0.672 | 0.539 | 0.048 | 0.019 |
| CLASS2_F0.01 | 18 946 | 0.827 | 0.105 | 0.704 | 0.501 | 0.055 | 0.019 |
| Cufflinks | 16 163 | 0.788 | 0.118 | 0.630 | 0.527 | 0.069 | 0.029 |
| IsoCEM | 21 906 | 0.597 | 0.066 | 0.525 | 0.325 | 0.017 | 0.005 |
| Scripture | 38 484 | 0.551 | 0.035 | 0.553 | 0.196 | 0.033 | 0.006 |
| iReckon | 30 180 | 0.591 | 0.052 | 0.611 | 0.290 | 0.262 | 0.061 |
| SLIDE | 72 867 | 0.841 | 0.028 | 0.787 | 0.148 | 0.243 | 0.023 |

*(Simulated transcripts were divided into high-, medium- and low-abundance classes based on their FluxSimulator-generated abundance. R = TP/(TP+FN) values were calculated within each class, and P = TP/#Transcripts values were based on the full set of transcripts.)*




| Program | $R^2$ |
|---|---|
| cufflinks | 0.804 |
| isoCEM | 0.977 |
| iReckon | 0.708 |
| SLIDE | 0.878 |
| Scripture | 0.674 |
| CLASS2 | 0.972 |
| CLASS2_F0.01 | 0.972 |

*Figure 1-4 Correlation of predicted and sampled ('truth') expression values for transcripts*

*(Left: Scatterplot of FluxSimulator-generated expression levels ('truth') and abundance values estimated by CLASS2. Right: Correlations between sampled expression levels and abundance estimates by each of the tested programs. Fully-reconstructed transcripts by each program, which could be unambiguously associated with sampled (reference) transcripts were included. $R^2$ = Pearson correlation.)*

### *Performance of programs in detecting alternative splicing events*

Even with the best data, predicting full-length splice variant transcripts from short RNA-seq reads aligned to the genome is prone to assembly errors. Alternative splicing events, which can be determined from the local structure of transcripts or reads, can be detected with more accuracy and are frequently used in studies [23, 24, 25]. We therefore analyze the ability of the programs to capture primitive classes of alternative splicing events, including exon skipping, intron retention and alternative exon ends. Since most programs do not specifically predict alternative transcription start and termination, we did not include them in the analysis. We compared events detected from transcripts generated by each of the programs to the set of events represented in the simulation data.

As Figure 1-5 indicates, CLASS2_F0.01 and Scripture are the best overall performers as indicated by their F-values, albeit the two programs have strikingly different behavior. Scripture captures the largest number of events in each category, but it does so at the expense of reporting a very large number of false positives, which can severely impact the significance of downstream analyses. CLASS2 and CLASS2_F0.01 find a large portion of the events in each category, balancing sensitivity with high accuracy and achieving the best tradeoff. More specifically, CLASS2 finds 25–36% more events in each category compared to Cufflinks, which is the leading reference annotation tool and is also the most precise of the programs, at higher or comparable precision. Moreover, CLASS2_F0.01 finds roughly twice as many events as Cufflinks in each category with only a relatively small drop in precision (4–17%). Like CLASS2, Cufflinks allows users to vary the stringency of the program. We therefore separately compared the performance when varying the parameter range of both CLASS2 and Cufflinks to control the

number of isoforms reported ('-F f', with f = 0.01, 0.02, 0.03, 0.05, 0.1, 0.1, 0.15). Cufflinks' performance dropped sharply from its default settings, whereas CLASS2 showed a consistent performance (Figure 1-6). CLASS2 extended the sensitivity range and, for the same sensitivity level, it delivered significantly higher precision. Therefore, using CLASS2 in its various settings has the highest potential for applications that involve studies of alternative splicing variation.

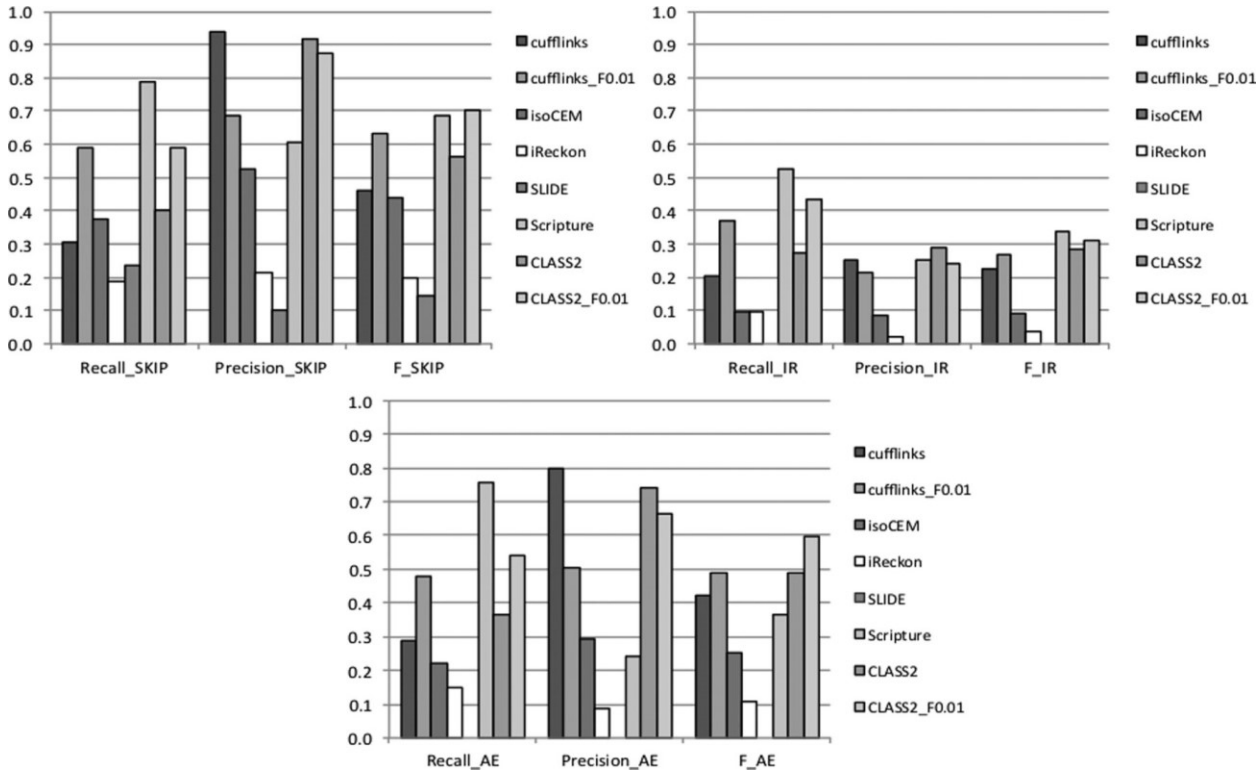



*Figure 1-5 Performance of programs in capturing alternative splicing events (Exon skipping (SKIP), intron retention (IR) and alternative exon ends (AE))*

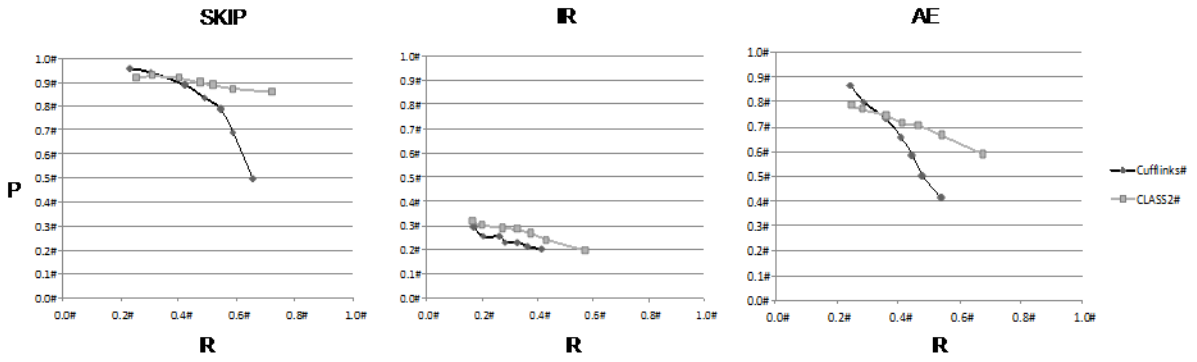



*Figure 1-6 Comparison of CLASS2 and Cufflinks in detecting alternative splicing events*

*(Recall (R, horizontal axis) and precision (P, vertical axis) of CLASS2 (light grey) and Cufflinks (dark grey) are shown for various stringency settings ('-F f', with f=0.01, 0.02, 0.03, 0.05, 0.1 and 0.2; right-to-left in the plots). The parameter 'F' controls the expression range of isoforms reported, as a fraction of the expression level of the most abundant isoform for the gene. Both programs have high specificity for stringent settings, but CLASS2 can detect more events overall and, for the same sensitivity (recall) level, has significantly higher precision than Cufflinks.)*

We next analyzed the errors made by these programs to evaluate their capacity to capture alternative splicing information. Programs detected exon skipping events with varying degrees of sensitivity (19–79%) and precision (10–94%). Notably, a majority of the false positives for all programs (67–86%; except for SLIDE, 23%) were matches to gene paralogs (Table 1-3), and only a small fraction were due to other alignment artifacts. This is most clearly illustrated by iReckon and IsoCEM, which predicted large numbers of splicing events, the majority of which were false positives. In contrast, most of the errors for SLIDE were due to spurious introns. The performance of all programs was significantly lower for intron retention events, with 10–52% sensitivity and only 2–29% precision. In most cases, false intron retention predictions resulted from mis-classification of 5' and 3' alternative gene starts and ends, as well as from cases in which a splice variant contained an exon that overlapped an intron in the corresponding gene (53–82% of false positives, except for Scripture, 23%). Lastly, programs in general were slightly

less accurate in capturing alternative exon ends compared to exon skipping, finding 15–76% of the true variations with 9–80% precision. The errors here were more evenly split between paralogs and variants present in the annotation but not sampled by the data (53–69%; except for iReckon 33%) and from spurious combinations of exon ends. CLASS2 had both a very low number and a very small percentage of false positives, matched only by Cufflinks, while detecting 30% more features (>90% more when CLASS2_F0.01 is used). These analyses also suggest that a simple way in which performance of most programs can be improved is by better distinguishing between true matches and paralogs, and that further improvements can come from better distinguishing between intron retention and other types of variation. Note that the simulated data does not model intronic reads resulted from unprocessed transcripts; the following sections provide a more realistic, albeit empirical, assessment on real data sets.

*Table 1-3 Programs' performance in capturing alternative splicing events*

| Program | Predicted | Correct | Recall | Precision | F-value | Artifacts | |
|---|---|---|---|---|---|---|---|
| Exon skipping (SKIP) | | | | | | Variants+ Paralogs | Spurious intron(s) |
| CLASS | 586 | 537 | 0.405 | 0.916 | 0.561 | 33 | 6 |
| CLASS_F0.01 | 897 | 783 | 0.590 | 0.873 | 0.704 | 92 | 9 |
| Cufflinks | 432 | 406 | 0.306 | 0.940 | 0.462 | 20 | 2 |
| Cufflinks_F0.01 | 1142 | 782 | 0.589 | 0.685 | 0.634 | 311 | 32 |
| IsoCEM | 940 | 496 | 0.374 | 0.528 | 0.438 | 380 | 33 |
| Scripture | 1724 | 1045 | 0.787 | 0.606 | 0.685 | 558 | 74 |
| iReckon | 1186 | 251 | 0.189 | 0.212 | 0.200 | 781 | 49 |

| SLIDE | 3022 | 311 | 0.234 | 0.103 | 0.143 | 618 | 2083 | |
|---|---|---|---|---|---|---|---|---|
| Intron retention (IR) | | | | | | Variants+ Paralogs | Spurious intron(s) | Mis-classified |
| CLASS | 176 | 51 | 0.276 | 0.290 | 0.283 | 17 | 12 | 52+41 |
| CLASS_F0.01 | 331 | 80 | 0.432 | 0.242 | 0.310 | 44 | 57 | 83+63 |
| Cufflinks | 150 | 38 | 0.205 | 0.253 | 0.227 | 19 | 13 | 43+50 |
| Cufflinks_F0.01 | 319 | 68 | 0.368 | 0.213 | 0.270 | 50 | 41 | 89+61 |
| IsoCEM | 205 | 18 | 0.097 | 0.088 | 0.092 | 25 | 61 | 48+51 |
| Scripture | 388 | 97 | 0.524 | 0.250 | 0.339 | 104 | 119 | 49+18 |
| iReckon | 818 | 18 | 0.097 | 0.022 | 0.036 | 116 | 56 | 392+204 |
| SLIDE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Alternative exon ends (AE) | | | | | | Variants+ Paralogs | Spurious intron(s) | Spurious combin. |
| CLASS | 496 | 369 | 0.363 | 0.744 | 0.488 | 62 | 4 | 61 |
| CLASS_F0.01 | 831 | 551 | 0.542 | 0.663 | 0.597 | 169 | 11 | 100 |
| Cufflinks | 367 | 293 | 0.288 | 0.798 | 0.424 | 39 | 5 | 30 |
| Cufflinks_F0.01 | 977 | 488 | 0.480 | 0.499 | 0.490 | 326 | 35 | 123 |
| IsoCEM | 761 | 223 | 0.219 | 0.293 | 0.251 | 372 | 40 | 126 |
| Scripture | 3,196 | 767 | 0.755 | 0.240 | 0.364 | 1656 | 197 | 576 |
| iReckon | 1,721 | 150 | 0.148 | 0.087 | 0.110 | 512 | 50 | 1009 |
| SLIDE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*(Programs were evaluated for their ability to detect 1327 exon skipping (SKIP), 185 intron retention (IR) and 1016 alternative exon end (AE) events present in the simulated data. Incorrect predictions were analyzed to determine classes of artifacts. Artifacts due to paralogs and splice*

*variants of the genes and transcripts in the sample were determined by comparison against events extracted from the full set of GENCODE annotations. The remaining events were searched for spurious introns and for class-specific error patterns, due to mis-classification of alternative first and terminal exons, or of reads from overlapping exons within the same or a different gene (IR), and to spurious combinations of exon start and exon end (AE).)*

### 1.3.2 Comparative evaluation on real data for different sequencing strategies

To assess the performance of programs on real data, we applied them to two large RNA-seq data sets. A lymphocyte sample from an individual free of neuropsychiatric disease was sequenced using two different library preparation strategies, as part of a twin study. In the first method, polyA-selected RNA was sequenced on an Illumina HiSeq2000 instrument to produce roughly 183 million 100 bp paired-end reads. This data set provides a good illustration of a typical RNA-seq analysis experiment, for which most programs are currently optimized. The second library was generated from the same lymphocyte sample by rRNA-depleting the total RNA, and sequenced to generate 317 million paired-end reads. Mapping all reads to the genome with Tophat2 produced roughly 170 million and 240 million read alignments, respectively, but comparatively a larger fraction (46% versus 7%) in the latter sample was in intronic reads.

#### *Comparison on the polyA-selected data set*

Because the current human genome annotation is inherently incomplete, while also including genes and isoforms not expressed in the sample, it is not possible to determine the true sensitivity and precision of any analysis tool on real data. Nevertheless, we deem consistency with the reference annotation, in particular for sensitivity, as a good indicator of a program's performance. Using a non-redundant set of GENCODE and RefSeq transcript models as reference, we compare the output of the six programs against the reference annotations.

Filtering out single exon assemblies, most of which are biological or computational artifacts, significantly increased the precision of Cufflinks and IsoCEM, whereas there was very little effect on the other programs.

Programs detected between 25–38% of the reference exons and 25–42% of the reference introns, but could only fully reconstruct a small fraction (4–9%; 7000–16 000) of the annotated transcripts. This is not unexpected, since only a subset of the reference annotations will be present in any given sample, but the small numbers make it difficult to differentiate among programs and determine the significance. To better assess the relative performance, we designate one method as reference and determine for each of the others the relative change in the number of transcripts found (Figure 1-7A, top). We chose Cufflinks as reference, because in our earlier testing on simulated data it was the most accurate among the reference programs.

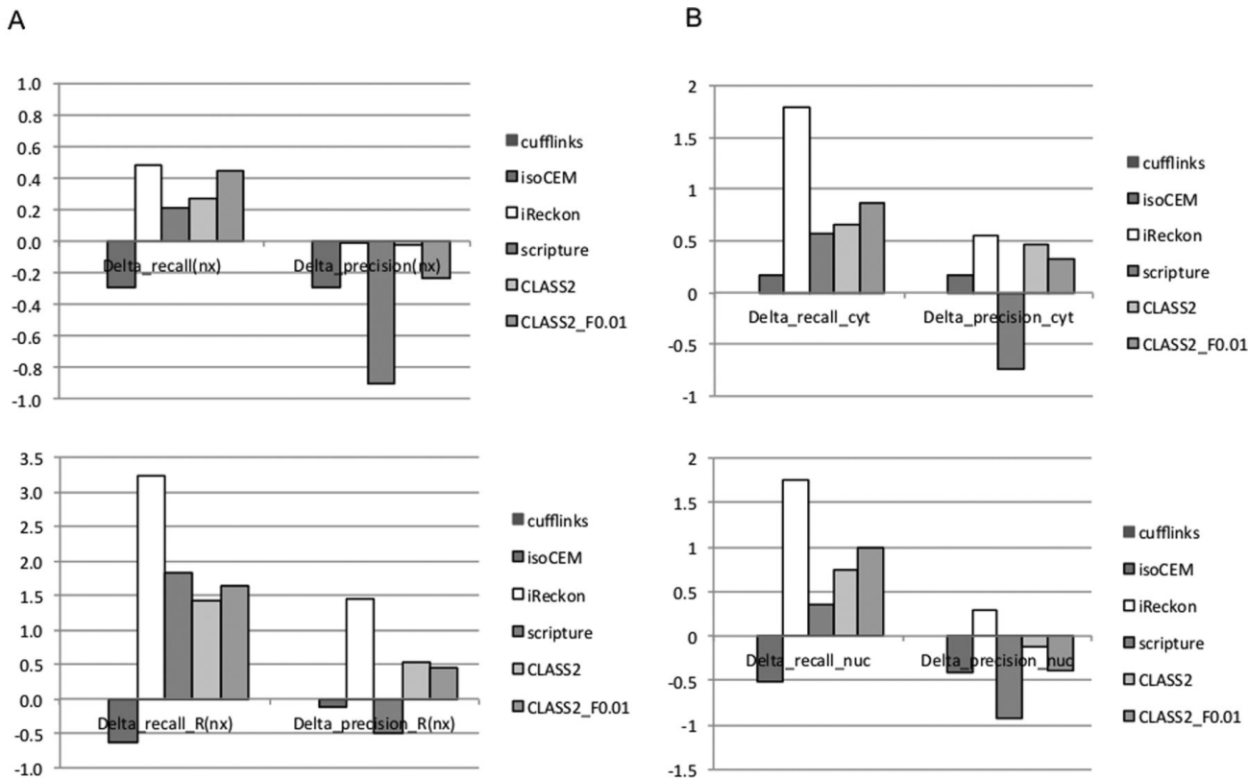



*Figure 1-7 Relative performance of programs on real data*

*(All values are relative to Cufflinks. (A) Performance on two real RNA-seq data sets from lymphocytes from the same individual: a polyA-selected data set (top), and an rRNA-depleted data set (bottom). (B) Performance with very deep sequencing data sets: the ENCODE IMR90 cell line, cytosol sample (top); same cell line, nucleus sample. For a program P, the relative performance improvement for recall is Delta_recall(P) = [TP(P) – TP(Cufflinks)]/TP(Cufflinks)], and similarly for precision. The value for Cufflinks (reference) is 0.)*

With the exception of isoCEM, programs find 21–48% more transcripts than Cufflinks, with iReckon and CLASS2_F0.01 reconstructing the largest numbers of reference transcripts. Cufflinks has the best precision again, followed very closely by iReckon and CLASS2. (Note that true 'precision' is impossible to assess, as 'false positives' could in fact represent true splice isoforms, not found in the reference annotation.) Overall, CLASS2 and CLASS2_F0.01 perform the best among *de novo* assemblers and offer the best tradeoff between sensitivity and precision, as measured by the F-value. When all programs are considered, iReckon appears to

perform the best; however, its performance is likely biased by the fact that it used as input the very set of gene annotations we now use for evaluation. When adjusting for paralog and spurious splice variant inflation, CLASS2 and CLASS2_F0.01 are the only two programs to exhibit positive cumulative gains in combined sensitivity and 'precision' (26% and 41%, respectively, more reference transcripts found compared to Cufflinks, at comparable or slightly lower precision). In conclusion, while Cufflinks appears to be the most precise of the programs for this type of data, CLASS2 is just as precise while more sensitive, and both CLASS2 and CLASS2_F0.01 offer more accuracy in combined sensitivity and precision.

### *Comparison on the rRNA-depleted data set*

We repeated the analysis on the rRNA-depleted RNA sample. Surprisingly, both Cufflinks and IsoCEM performed very poorly, finding only a small subset of reference features; we suspect the reason is that both employ a local intronic 'noise' filter at the individual intron level, whereas other programs characterize 'noise' at gene (iReckon, CLASS2) and/or genome level (Scripture, CLASS2). Rankings for other programs were similar to those for the polyA+ data (Figure 1-7A, bottom). Although this data set does not fit the characteristics of the simulated data, which was modeled after the polyA-selected RNA sample preparation, we again conjecture that a large portion of iReckon's performance is in fact due to over-counting of paralogs and alternative exon combinations toward the true matches. CLASS2 and CLASS2_F0.01 are robust with the intronic noise levels and produce reliable gene models, having the best accuracy among *de novo* assemblers. In particular, they can reconstruct 2.5 times as many transcripts as Cufflinks. An example illustrating the programs' performance at the UBR4-CAPZB locus is shown in Figure 1-8.

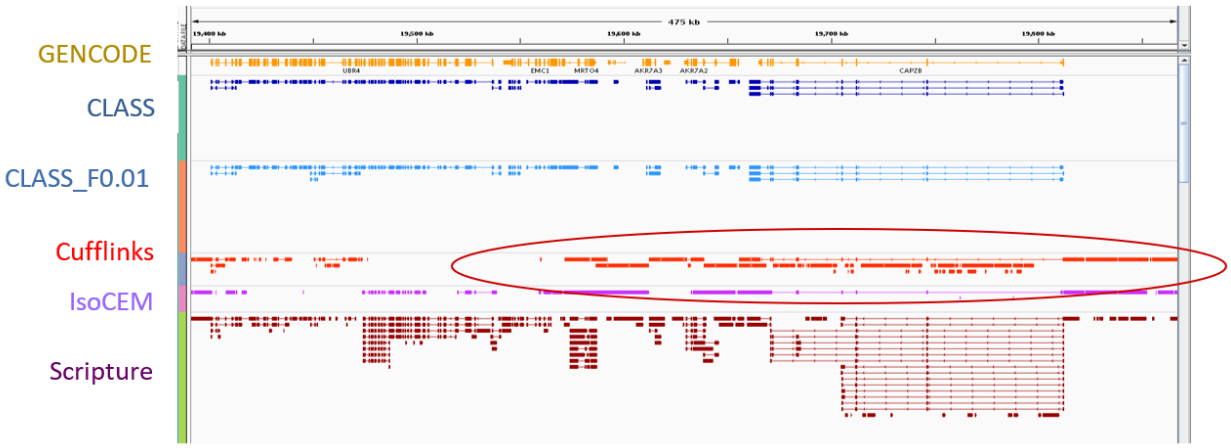



*Figure 1-8 Illustration of program output at the UBR4-CAPZB gene locus (lymphocyte, rRNA-depleted sample)*

*(IsoCEM and Cufflinks fail to identify full-length transcripts models, and are confounded by intronic noise (red circle). )*

### *Validation of predicted intron retention events*

Intron retention has recently been shown to play a part as a regulatory mechanism in cellular differentiation and tumor-suppressor inactivation [26, 27]. Yet, intron retention events are difficult to identify from RNA-seq data and are likely under-represented in the gene annotation databases. To illustrate the ability of CLASS2 to identify novel alternative splicing events, we performed PCR validation on two intron retention events detected from the PBL RNA-seq data by CLASS2 and were not found by any of the other programs: the 304 bp chr12:1 908 861–1 909 166 intron at the CACNA2D4 (Calcium channel voltage-dependent Alpha 2/Delta subunit 4) gene locus, and the 888 bp chr12:9 985 010–9 985 899 intron within the KLRF1 (Killer cell lectin-Like receptor subfamily F, member 1) gene. Human blood cDNA and genomic DNA were amplified with primer sets targeting intron retention events in the two genes (Figure 1-9). The primers were designed to span a nearby exon–intron junction to demonstrate the intron retention event occurred in a spliced mRNA transcript. Strong PCR products of the expected

size were observed in cDNA but not genomic DNA. The cDNA PCR product was then cloned and sequenced to demonstrate the retention of the intron. Integration with other data sets in the UCSC Genome Browser shows supporting evidence from one mRNA (accession: BX537436) for CACNA2D4, but no evidence was previously available for KLRF1, therefore demonstrating the power of the approach.

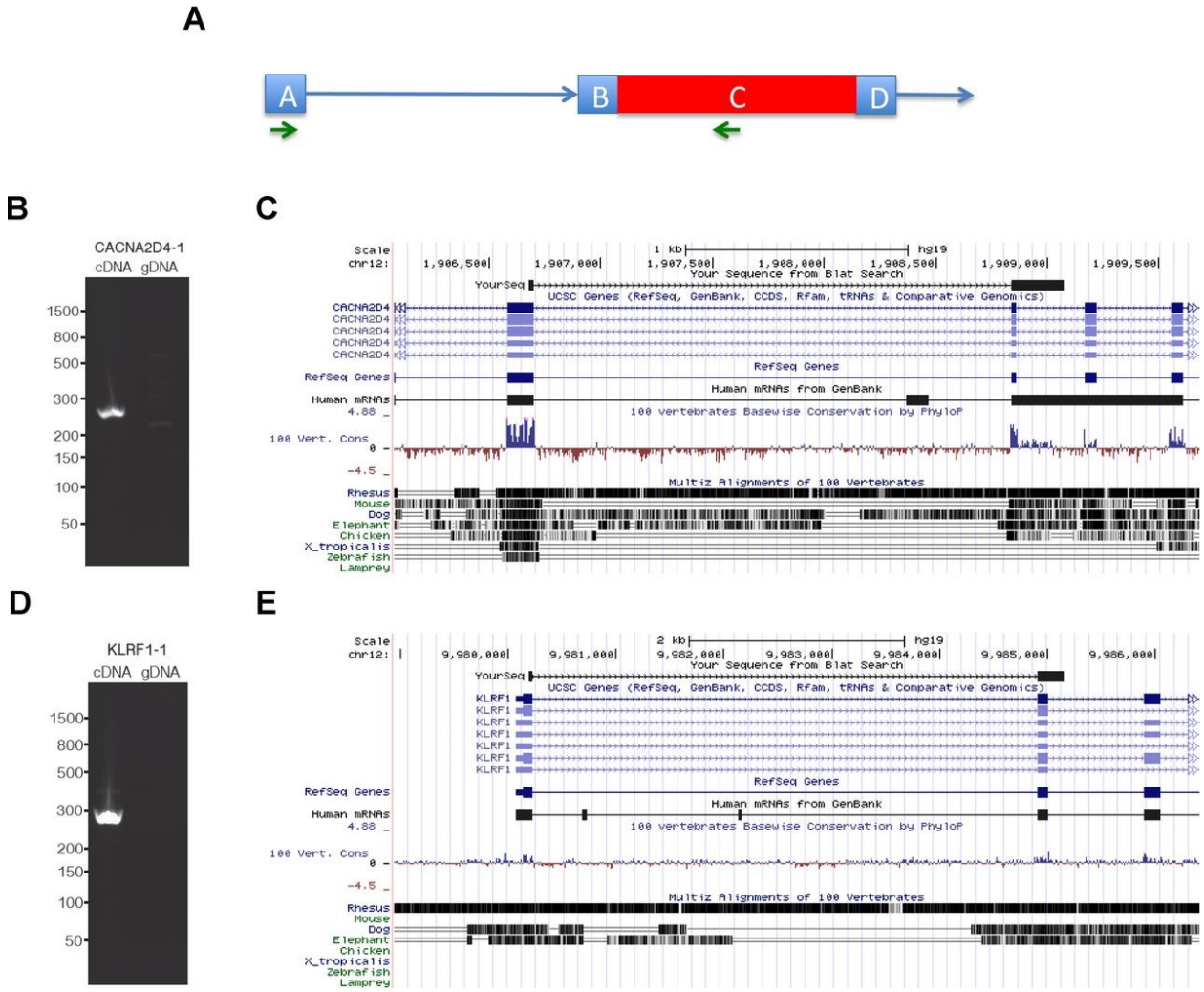



*Figure 1-9 PCR validation of CLASS2 output*

*((A) PCR validation strategy: blue squares represent annotated exons, the red rectangle represents the identified intron retention event, and the blue lines with arrowheads represent introns. Green arrows denote the location of the PCR primers. Human blood cDNA and genomic*

*DNA were amplified with primer sets targeting intron retention events in (B) CACNA2D4 and (D) KLRF1 genes. For each primer set, a strong PCR product of the expected size was observed in cDNA but not genomic DNA. The sequences of the PCR reactions for (C) CACNA2D4 and (E) KLRF1, labeled 'YourSeq' in the figure, were aligned against the human genome using the UCSC Genome Browser.)*

### *Performance of programs on very deep sequencing data sets*

The fast and cost-effective RNA-seq technology has led to a steady increase in the data size and depth of sequencing, enabling detailed alternative splicing studies. To tackle very large data sets, some programs focus on determining the major isoforms and therefore provide a limited view of the splicing repertoire in a sample, whereas others simply cannot handle the combinatorial explosion. To assess the potential for discovering splicing variation from deep sequencing data sets, we applied all programs to two very large data sets produced by the ENCODE project [28, 29]. The IMR90 lung fibroblast cell line was sequenced at great depth in three separate surveys, of the whole cell, the cytosolic and the nuclear fractions. Two replicates were run for each fraction, which can be used in our evaluation to assess the accuracy of the predicted features by testing their reproducibility in multiple samples. To reduce the run time, below we restricted our accuracy analyses to chromosome 1. Even so, SLIDE was prohibitively slow and was excluded from the analysis. Summary results of programs are listed separately (Table 1-4).

With >300 million reads, the ENCODE IMR90 data sets are among the most deeply sequenced to date and are expected to sample RNA biotypes not found in the reference annotations. Therefore, true accuracy (especially precision) is not possible to assess since novel splice variants will be counted as false positives. Nevertheless, we again judge concordance with annotated features (introns and full transcripts) as indicative of sensitivity and leverage the

reproducibility of features across the six samples to better estimate the programs' performance.

When considering the goal of reconstructing full transcripts, iReckon has seemingly the best performance, as it identified the largest number of transcripts present in the existing annotations (Figure 1-7B). Again, however, these results should be considered with caution given the large inflation from variants and paralogs observed with simulated data. Excluding iReckon, both CLASS2 and CLASS2_F0.01 reconstruct the largest number of annotated transcripts in both the cytosol and the nucleus samples, 60–90% (77–103% nucleus) more than Cufflinks and 15–43% (30–49% nucleus) more than the best of the programs, while also having higher or comparable 'precision'.

We separately evaluated the programs' accuracy in capturing deeper splicing variation, in particular novel variation, using splice junctions (introns) as surrogates (Table 1-4). CLASS2 and CLASS2_F0.01 find by far the most known introns, 8% and 11% more than the best of the other programs on the cytosolic sample, and 22% and 37% more on the nucleus sample. When including in the reference those novel introns that are reproducible in at least two data sets, CLASS2_F0.01 remained the most sensitive, followed by CLASS2 and Scripture, at very high precision (>97% for cytosol and >95% for nucleus).

*Table 1-4 Performance of programs on the ENCODE IMR90 data*

| Program | Transcripts | | | | Introns | | | |
|---|---|---|---|---|---|---|---|---|
| | Predicted | Match | R | P | Predicted | Match | R | P |
| Cytosol | | | | | | | | |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| CLASS2 | 3029 | 1053 | 0.068 | 0.348 | 12 662 | 12 557 (11 996) | 0.378 (0.361) | 0.968 (0.947) |
| CLASS2_F0.01 | 3836 | 1183 | 0.076 | 0.308 | 13 413 | 13 253 (12 327) | 0.399 (0.371) | 0.988 (0.919) |
| Cufflinks | 2508 | 621 | 0.040 | 0.248 | 10 420 | 10 372 (10 109) | 0.312 (0.304) | 0.995 (0.970) |
| Cufflinks_F0.01 | 3458 | 719 | 0.046 | 0.208 | 11 725 | 11 564 (10 779) | 0.348 (0.325) | 0.986 (0.919) |
| IsoCEM | 2479 | 722 | 0.047 | 0.291 | 11 483 | 11 297 (10 617) | 0.340 (0.320) | 0.984 (0.925) |
| Scripture | 14 621 | 971 | 0.063 | 0.066 | 13 820 | 12 751 (11 149) | 0.384 (0.336) | 0.923 (0.807) |
| iReckon | 4512 | 1730 | 0.112 | 0.383 | 11 724 | 11 477 (10 552) | 0.346 (0.318) | 0.979 (0.900) |
| Nucleus | | | | | | | | |
| CLASS2 | 6084 | 992 | 0.064 | 0.163 | 16 391 | 15 765 (12 862) | 0.475 (0.418) | 0.962 (0.846) |
| CLASS2_F0.01 | 10 216 | 1141 | 0.074 | 0.11 | 18 610 | 17 699 (14 539) | 0.532 (0.438) | 0.950 (0.781) |
| Cufflinks | 2714 | 561 | 0.036 | 0.207 | 11 255 | 11 079 (10 576) | 0.334 (0.319) | 0.984 (0.940) |
| Cufflinks_F0.01 | 6085 | 789 | 0.051 | 0.13 | 16 884 | 16 064 (13 568) | 0.484 (0.409) | 0.951 (0.804) |
| IsoCEM | 2236 | 277 | 0.018 | 0.124 | 9604 | 8737 (7576) | 0.263 (0.228) | 0.910 (0.789) |
| Scripture | 45 247 | 764 | 0.049 | 0.017 | 18 048 | 13 910 (10 188) | 0.419 (0.307) | 0.771 (0.564) |
| iReckon | 5769 | 1539 | 0.099 | 0.267 | 10 162 | 9474 (8232) | 0.285 (0.248) | 0.932 (0.810) |

*(Features (full-transcripts and introns) matching known and/or high-confidence novel annotations. GENCODE v.17 chromosome 1 annotation contains 15 493 transcripts and 33 202 introns. R = (recall) = Match/Annotations, P = (precision) = Match/Predicted.)*

Lastly, CLASS2 completed the task in roughly 30 min for the chromosome 1 of the cytosol sample and was comparable in speed with the fastest of the programs (Table 1-5). As a practical matter, for increased efficiency CLASS2_F0.01 can be run first to report a comprehensive set of transcripts, and the output can be filtered using various '-F' parameters (minimum fraction of reported isoforms' abundance from that of the most expressed isoform) to produce increasingly more precise subsets, at the cost of finding fewer transcripts. Therefore, results for CLASS2 with multiple settings can be obtained in roughly the same time as a single run.

*Table 1-5 Running times of transcript assembly algorithms*

| Program/ Data set | Time (wall clock) | | | | |
|---|---|---|---|---|---|
| | Sim | PBL polyA+ | PBL rRNA- | IMR90 Cytosol (chr1) | IMR90 Nucleus (chr1) |
| CLASS2 | 390m | 296m | 244m | 31m | 747m |
| Cufflinks | 488m | 1679m | 4245m | 84m | 548m |
| isoCEM | 168m | 169m | 225m | 18m | 67m |
| iReckon | 6097m | 5696m | 17481m | Na | Na |
| SLIDE | ~1.5 weeks | Na | Na | Na | Na |
| Scripture | 1039m | 1262m | 1574m | 27m | 75m |

*(Run times for CLASS2 and CLASS2_F0.01 are largely the same, since the algorithm first detects a comprehensive set of transcripts and then applies the expression cutoff to select a subset. Run times for iReckon included the time for internally re-mapping the reads to the genome with bwa, and were excluded. All times measured on a Unix machine with 512 GB RAM and 2100 MHz CPU, single-threaded. Memory usage for CLASS2 for all tasks was <3 GB RAM. PBL = Peripheral Blood Lymphocytes.)*

### *De novo annotation of a newly sequenced organism*

Next generation sequencing has significantly accelerated the pace at which new genomes are being produced. Annotation projects for these genomes are increasingly relying on fast and low cost RNA-seq resources. The choice of RNA-seq transcript assembler here is critical; for instance, since annotation-based programs are not designed to identify novel genes, *de novo* methods are the most productive. To illustrate CLASS2's ability to annotate new genomes, we apply it to enhance the annotation of the peach genome. With its 226.6 MB of sequence assembled in 365 scaffolds, the Prunus persica (peach) genome is a good model for future plant species annotation projects. We use CLASS2 to analyze four RNA-seq data sets sampled from embryo and cotyledon, fruit, root and leaf of peach tree (PRJNA34817), totaling 164.1 million 75 bp paired-end reads. Preliminary gene annotations are also available, and we use them to identify novel transcript variants that could be used to enhance the existing annotation.

Following read mapping and assembly, CLASS2 produced between 15 000–27 500 transcript fragments (transfrags) per sample (Table 1-6). When compared across the four samples, these amounted to roughly 19 500 transfrags corresponding to existing annotations, but also more than 1000 new loci, each present in at least two of the samples, and 27 161 novel transcripts of known genes, representing new splice variants or extensions of the annotated transcripts. In one example at the ppa023343m gene locus (Figure 1-10A), transfrags assembled from short reads extended the existing gene model by 10–11 exons and revealed several novel splice variations. The extended gene encodes a 1016 aa protein that has similarity over its entire length to importin-11 and importin-11-like proteins in other species (Prunus mume, Vitis vinifera, Citrus simensis, Fragaria vesca, Theobroma cacao and Glycine max). In another

example at the ppa023750 gene locus, transfrags assembled from the four RNA-seq samples point to additional splice variants, including a novel skipping event of a 39 bp exon located at scaffold_1:4613746–4613784, and a potential retention of an 84 bp intron (scaffold_1:4621242–4621327; Figure 1-10B), manifested only in the embryo and cotyledon sample. The landscape for this gene is also significantly reconfigured, by merging two previously adjacent genes and by a further extension of its 5' end. The gene has extensive and close similarity to predicted proteins in apple, Japanese apricot, orange, and cacao. Lastly, a new gene locus, located between genes ppa026188m and ppa005862m, and several putative splice variants discovered with CLASS2 can be seen in Figure 1-10C. Blast searches of the two novel putative gene sequences found distant homologs elsewhere in the genome, as well as matches to cytochrome C oxidase subunit 6b protein and to predicted FLX-like proteins in several Rosaceae species. Both sequences contain long open reading frames (762 bp out of 1347 bp, and 234 bp out of the 366 bp sequences, respectively) and are strong candidates for novel, not yet annotated genes.

*Table 1-6 Annotation of a newly sequenced organism (peach)*

| Set | Reads | Mapped | Genes | Transcripts | Genes | Transcripts |
|---|---|---|---|---|---|---|
| | | | (CLASS2) | (CLASS2) | (CLASS2_F0.01) | (CLASS2_F0.01) |
| SRR531862 | 42394368 | 4E+07 | 17320 | 22617 | 17,322 | 27,442 |
| SRR531863 | 41589898 | 3E+07 | 16313 | 20799 | 16,313 | 24,614 |
| SRR531864 | 42341754 | 3E+07 | 16320 | 18,397 | 16,321 | 19,816 |
| SRR531865 | 38883238 | 2E+07 | 12083 | 13,752 | 12,083 | 14,935 |

*(Summary of mapping and assembly results are shown for the four RNA-seq samples (SRR531862 – embryos and cotyledons, SRR531863 – root, SRR531864 – fruit and SRR531865 – leaf). 'Mapped' represents the number of reads mapped with 10 or fewer matches on the peach genome. The last two columns give the numbers of loci ('genes') and transcripts assembled with CLASS2, using the default and the sensitive ('-F 0.01') settings.)*

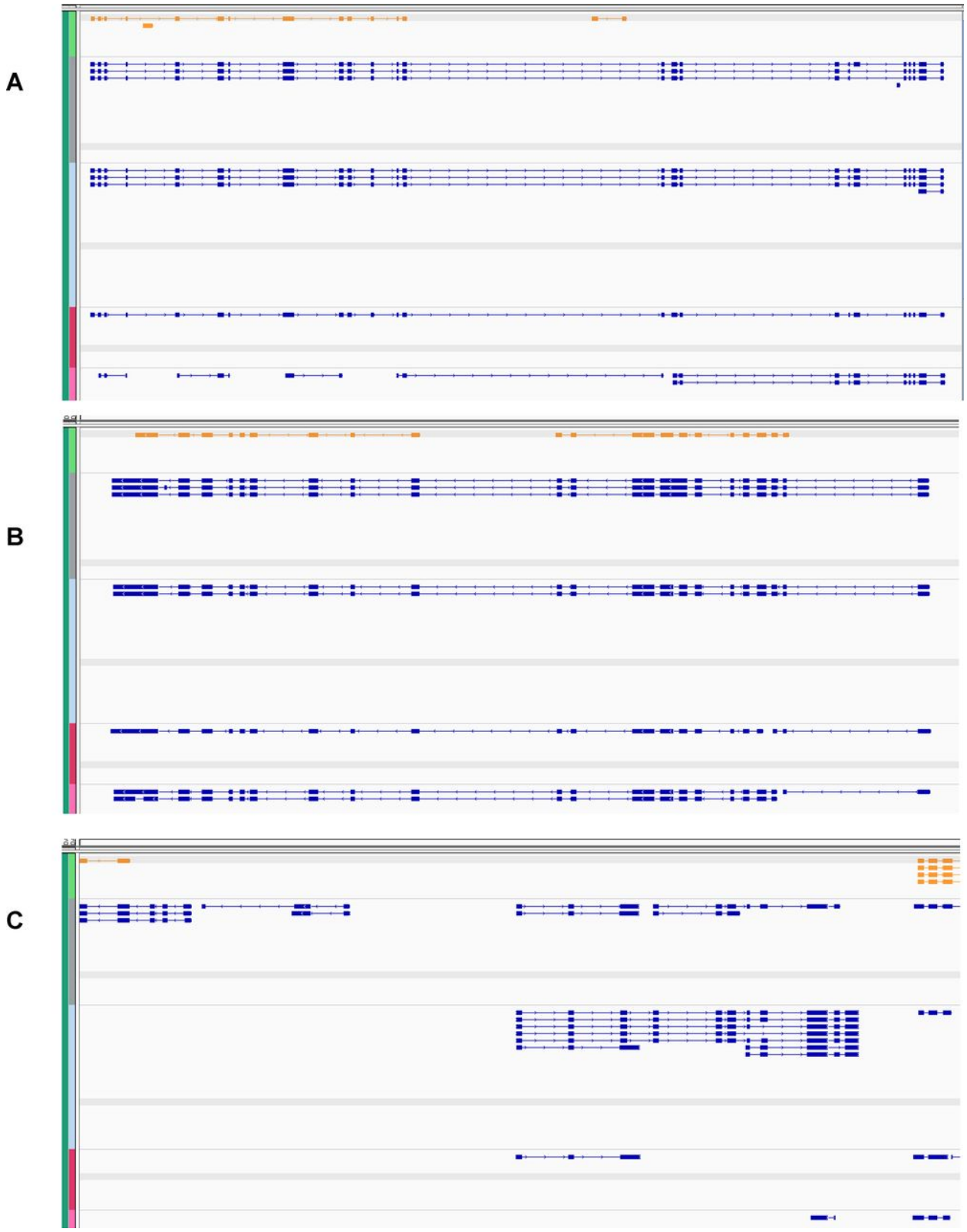



*Figure 1-10 Refining the peach gene models*

*(CLASS2 transcript predictions for four peach RNA-seq data sets (BioProject ID: PRJNA34817) are shown in blue, and reference annotations in gold. (**A**) RNA-seq reads assembled with CLASS2 extend the ppa023342m gene model by 10–11 exons and suggest additional splice variants. The extended gene model is supported by data in all of the four samples. (**B**) An extended gene model and several novel splice variants at the ppa023750m gene locus. The intron bridging the two existing gene annotations has (18,7,9,8) supporting reads, respectively, in the four samples, and the last intron is supported by (8,15,6,9) reads. Further, the 39 bp novel exon at scaffold_1:4613746–4613784 in the SRR531862 sample is alternatively skipped in the reference annotation, and there is ample intronic read support for a putative 84 bp frame-preserving intron retention event at scaffold_1: 4621242–4621327. (**C**) CLASS2 finds novel genes and splice variants in the intergenic region between annotations ppa026188m and ppa005862m.)*

## 1.4 Conclusions

A wealth of RNA-seq data, from small individual projects to very large-scale systematic experiments, is making it possible for the first time to catalog alternative splicing variation in detail in different organisms, tissues, at various developmental stages and stress or disease states, and in individual cell types. Many computational methods have already been developed to translate the data into knowledge at the level of genes and transcripts. However, they are still far from being able to assemble full transcript models with high accuracy [30] and have limited ability to capture even local splicing variation, including canonical alternative splicing events. Some classes of events are especially difficult to detect due to artifacts that occur during data generation and mapping (Figure 1-5), and have not been systematically pursued by current programs.

We developed a novel splice graph-based algorithm and software tool, CLASS2, with the goal to assemble likely models of full-length transcripts while capturing local splicing variations with high accuracy, to allow genome and system-wide alternative splicing analyses. CLASS2 employs intronic reads and splice junction ‘noise’ models to accurately determine the set of parts, namely exons and introns, and a novel time and memory efficient dynamic programming

algorithm to select a subset of probable transcripts that retain most of the splicing variation in the sample.

CLASS2 differs technically from existing approaches while promoting alternative splicing discovery in several ways: (i) it uses an LP-based system to locally predict exon variations, such as alternative 5' and 3' exon ends; (ii) it incorporates a combined gene- and genome-level model of intronic 'noise', to distinguish retained introns; (iii) it models alternative first and last exons, including the cases when they occur at internal exons; and (iv) it uses an iterative algorithm and a complex scoring system to select a minimal subset of transcripts that collectively retain as much splicing variation as possible while explaining all the reads.

CLASS2 also implements several memory and time saving strategies that are critical to its performance and allow it to run on very deep sequencing data sets without sacrificing accuracy. These include a smaller LP system formulated on gene regions rather than along the entire gene, which is both faster and more accurate to solve; clustering reads into classes ('constraints'); employing a compact and scalable splice graph representation of genes; and, last but not least, implementing a new dynamic programming transcript selection algorithm that avoids enumerating transcripts in complicated graphs, and is memory and space efficient. As a result, a typical run on an Illumina-generated 200 million paired-end read set requires less than 3 GB RAM and, when run with multiple threads, takes only a few hours and therefore can be run on most desktop computers.

In our comparative evaluation of CLASS2 and several state-of-the-art programs, we found CLASS2 to be significantly more sensitive in capturing alternative splicing variations, at both the

level of full transcripts and for local alternative splicing events, at precision higher or comparable with that of the best program. In particular, it detected almost twice as much variation as Cufflinks, the most precise of the programs, with only a small decrease in precision. The evaluation also afforded us a unique view of the strengths and limitations of the different approaches. For instance, annotation based approaches as employed by SLIDE and iReckon can detect a larger number of the reference annotations, but are also prone to reporting paralogs and splice variants not actually present in the sample. This is particularly problematic when interpreting the programs' output on real data, where they would be incorrectly labeled as true matches. The quantity and quality of data can create significant challenges, while library sample preparation can further introduce biases and significantly alter the characteristics of the data [31]. In general, we found Cufflinks to be the most precise of the programs but missing important splice variations, and Scripture to be the most sensitive but imprecise. However, while different programs may score best by various criteria and for different types of applications, CLASS2 delivered a consistently good performance for a wide variety of applications and sequencing strategies. These included surveys of polyA-selected (spliced) RNA, which are the most frequent among RNA-seq applications, as well as of ribosomal depleted total RNA, and very deep sequencing experiments to characterize splicing variation, low expression forms, and novel and cellular fraction-specific RNA biospecies, in great depth.

While the boundary between true and noisy splice variation [32] continues to remain undefined, making it ever more difficult to determine the extent of splicing variation and number of isoforms for any given gene, some strategies could help improve the outcome. Better methods are needed to characterize the various types of artifacts that confound classes

of variations, such as alternative polyadenylation or alternative promoter usage and retained introns. These can entail implementing sequence models of binding sites of regulatory proteins [33, 34], or incorporating other types of evidence including CAGE tags, DNase-seq or FAIRE-seq signals, paired-end diTags (PET-seq) [35] and polyA-seq [36] sequences, where available. Also needed are complete reference data sets on genes or systems that can help evaluate the performance in an unbiased way, or at the very least better simulation models. The latter should include realistic models for sequencing artifacts, including intronic reads from unprocessed pre-mRNA, as well as for the amount and complexity of splicing variation with increasing sequencing depths, and for different types of RNA-seq experiments. Even further, accuracy measures are needed to be able to evaluate programs for their ability to reconstruct splice variations at both global and local levels, including canonical alternative splicing events and local assemblies. Current evaluation schemes focus on the reconstruction of full-transcripts, discounting correct partial reconstructions. Lastly, new sequencing technologies or continuous improvements in the existing ones that extend both read and insert lengths will provide increasing contiguity, while large and judiciously designed experiments will provide multiple replicates or concordant data sets that can be analyzed simultaneously [37, 38] to improve both throughput and accuracy.

Work on this project was supported in part by NSF award ABI-1159078, ABI-1356078 and IOS-1339134 to L.F., Stanley Medical Research Institute to S.S.. CLASS and CLASS2 are available free of charge for all and under a GNU GPL license from http://sourceforge.net/projects/Splicebox.

# Chapter 2

# PsiCLASS: efficient and scalable transcriptome assembly from multiple RNA-seq samples

## 2.1 Introduction

RNA sequencing has become the *de facto* standard in surveying the transcriptome of a cell, organism or species, to determine the expressed genes and transcripts and their expression levels, and to enable differential and functional analyses [39, 7]. A crucial step in virtually all RNA sequencing (RNA-seq) data analyses is assembling the reads into full-length transcripts. The accuracy of transcript reconstruction is critical for quantification, detection and characterization of alternative splice variants, and the identification of differences in gene expression and splicing patterns between tissues, developmental stages, and physiological or disease states.

Virtually all transcriptomic studies involve multiple samples. The current paradigm is to assemble the reads in each sample, then merge the partial transcripts (transfrags) across all samples to create a unified set of meta-annotations [40], which is used as reference for downstream quantification and differential analyses. Most single-sample assemblers including Cufflinks [9], isoCEM [11], Scripture [10], Traph [41], CLASS [17], iReckon [12], CIDANE [42], FlipFlop [43], CLASS2 [44], StringTie [45], Scallop [46] and TransComb [47] build a graph

structure from read alignments on the genome, then traverse the graph to select an optimized set of transcripts, represented as paths. Recent transcript assembly methods including StringTie, CLASS2 and Scallop have taken great strides towards increasing the accuracy and efficiency of assembly at single-sample level, and meta-assemblers such as StringTie(ST)-merge and TACO [40] have led to more robust collections of meta-annotations. Despite these efforts precision remains low, with less than 40% of the predicted transcripts in a single-sample and less than 30% of transcripts in meta-annotations representing complete and accurate reconstructions [40, 30].

We present PsiCLASS, based on a novel approach that simultaneously analyzes multiple RNA-seq samples, which achieves significantly higher precision at sensitivity comparable to the best current approaches, and significantly higher overall accuracy in its default setting. PsiCLASS is a combined assembler and meta-assembler: it reports a set of transcripts for each sample, as well as a set of meta-annotations obtained by combining the individual samples' outputs. PsiCLASS starts by selecting a set of high-confidence introns and subexons at each locus, using novel statistical models of introns and intronic read levels. It then builds a unified subexon splice graph that is used within a dynamic programming optimization procedure to select a representative set of transcripts in each sample (See 2.2 and Figure 2-1 for details). Lastly, PsiCLASS extracts a subset of meta-annotations from the aggregated transcript sets by voting.

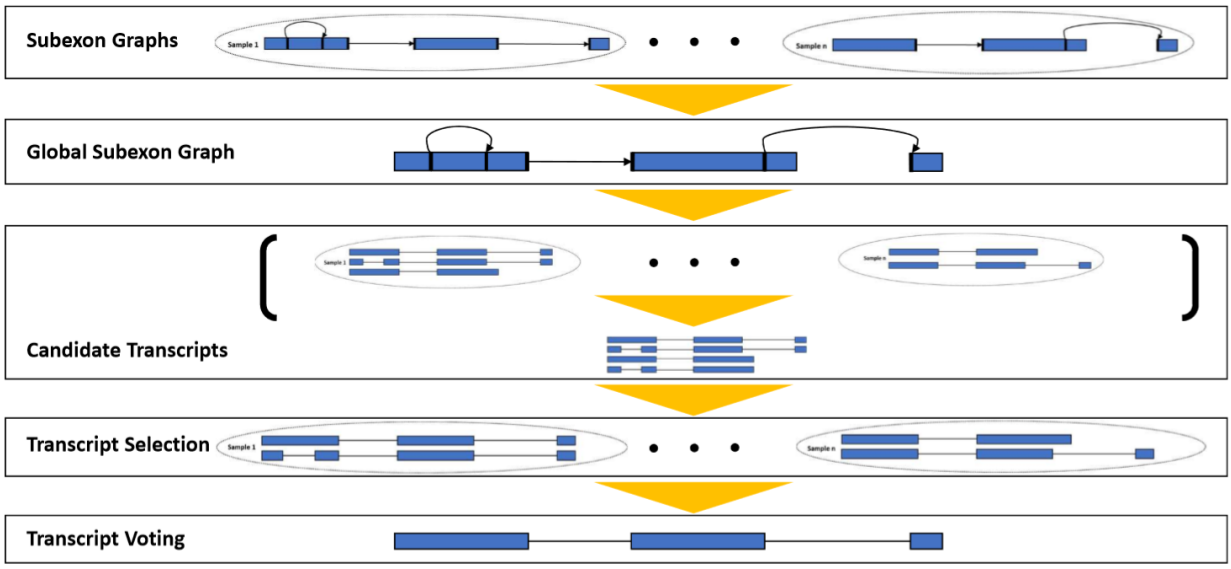



*Figure 2-1 Overview of the PsiCLASS algorithm*

*(Step 1. Build sample-level subexon graphs from aligned reads and splice reads. PsiCLASS builds a subexon graph for each sample by clustering overlapping read alignments into regions, dividing regions into subexons at splice junctions (inferred from spliced reads), and connecting with edges subexons that are adjacent within the same region or connected by an intron. Step 2. Build and refine a global subexon graph, by merging sample-level subexon graphs and employing intron and subexon filters that evaluate information simultaneously across all samples. Step 3. Enumerate or select a set of candidate transcripts using dynamic programming across all samples. Step 4. Select a subset of transcripts in each sample, using a greedy strategy that iteratively select an optimal transcript (with global subexon graph-based dynamic programming). Step 5. Select a unified set of meta-annotations from among the sample-level transcripts, with voting.)*

## 2.2 Methods

### 2.2.1 Algorithm overview

PsiCLASS builds a global subexon graph of a gene and its splice variants from genome-aligned RNA-seq reads in all input samples. It then traverses the graph to select a subset of the encoded transcripts in each sample. Lastly, it combines the predicted transcript sets across all samples, using a voting procedure to select a final set of meta-annotations.

### 2.2.2 Building per sample subexon graphs

PsiCLASS builds a subexon graph for each sample, then combines graphs across all samples to create a global subexon graph.

In each sample, PsiCLASS uses candidate introns extracted from spliced read alignments to divide the genome into regions and subexons. A region denotes a maximal contiguous portion of the genome covered by reads. A subexon is a portion of a region delimited by two consecutive splice junctions and/or the end(s) of the region. A subexon graph has subexons as vertices, and two subexons are connected by an edge if they are adjacent in the same region or connected by an intron. Candidate splice variants are encoded as maximal paths in the subexon graph.

To build the sample-level subexon graph, PsiCLASS clusters read alignments along the genome that are co-located and on the same strand. *Introns* are extracted from spliced alignments and used to divide the region into subexons. A major confounding factor in determining subexons from RNA-seq data is the presence of intronic unprocessed RNA ('noise'). To differentiate between intronic 'noise' and 'signal', such as retained introns or alternative 5' and 3' gene ends, PsiCLASS assigns each subexon a score that reflects the probability that it is 'noise'. In contrast to current single-sample methods, which simply discard a subexon if it fails sample-wide cutoffs, PsiCLASS then combines sample-level scores across all samples to determine a final 'label' for the subexon and its inclusion in the global subexon graph.

More specifically, PsiCLASS computes the probability that a subexon is due to intronic 'noise' using two models: i) the exon-intron coverage ratio, and ii) the intronic read coverage. Let $c_i$ be

the average read coverage of (intronic) subexon i. In the *coverage ratio* model, PsiCLASS calculates a score that is equal to the coverage ratio of this subexon versus its flanking subexons: $r_i = \min(\frac{c_i}{c_{i-1}}, \frac{c_i}{c_{i+1}})$. The score is fitted to a mixture of two Gamma distributions, one representing 'signal' and one 'noise': $\mathrm{p(r_i)} = \pi Gamma_{\theta_0,k_0}(r_i) + (1-\pi)Gamma_{\theta_1,k_1}(r_i)$, where $\pi$, $(1-\pi)$ are the prior probabilities that an intronic subexon is 'noise' or 'signal', respectively, and $\theta_0, k_0, \theta_1, k_1$ are the parameters for the Gamma distributions, calculated with an expectation maximization (EM) algorithm. With these parameters, PsiCLASS can infer the probability that subexon i is 'noise' according to the Bayes formula: $\mathrm{P_R(r_i)} = \frac{\pi Gamma_{\theta_0,k_0}(r_i)}{\pi Gamma_{\theta_0,k_0}(r_i)+(1-\pi)Gamma_{\theta_1,k_1}(r_i)}$.

The coverage ratio model above is insufficient when the overall gene coverage is low. Hence, the second model establishes a similar formula for coverage levels, $\mathrm{P_C(c_i)}$, with $\theta'_0, k'_0, \theta'_1, k'_1$ the parameters inferred using coverage. The final per sample subexon score then is $\mathrm{P}(i) = \max(P_R(r_i), P_C(c_i))$.

### 2.2.3 Building the global subexon graph

PsiCLASS removes likely artifactual introns and intronic 'noise' subexons by evaluating evidence across all sample, and builds a global subexon graph by combining individual samples' graphs that share at least one intron.

*Multi-sample intron selection.* To select a highly accurate set of introns, PsiCLASS assesses each candidate intron's read support across all samples. Assume the experiment contains *M* samples, and denote each intron by its coordinates in the genome, e.g. $(a, b)$. Let $S(a, b)$

denote the total number of read alignments supporting $(a, b)$ over all samples. Then the total number of alignments supporting its splice sites: $S(a) = \sum_{(a,y)} S(a, y)$ , $S(b) = \sum_{(x,b)} S(x, b)$. PsiCLASS keeps intron $(a, b)$ iff: *i)* $\frac{S(a,b)}{M} \geq 0.5$, indicating strong read support in one or a few samples, or consistent read support across multiple samples; and *ii)* $(a, b)$ appears in at least M0 samples, where $\mathrm{M0} = \min(\left\lceil \frac{M}{50} \right\rceil (\lfloor \frac{b-a+1}{100{,}000} \rfloor + 1), M)$, if |b-a|>=100,000 (long intron). Condition ii) is intended to filter out long intron-type alignment artifacts due to gene families and repeats, or from sequencing errors, which can lead to merged genes and transcripts.

*Multi-sample subexon selection.* To determine a global set of subexons, PsiCLASS combines the subexon sets of individual samples with some modifications. Where multiple 3' or 5'-end (*i.e.*, subexons not delimited by a splice site) candidate subexons occur with the same endpoint and potentially different lengths among the samples, PsiCLASS creates a unique subexon with the median length. Further, to determine intronic subexons, PsiCLASS calculates a final score by combining all sample scores with a Bayesian formula. More specifically, let $\bar{\pi}$ denote the prior probability of intronic 'noise' in the global model, calculated as the average of the mixture coefficients of the samples. Then the subexon score: $\mathrm{P_n}(\mathrm{noise}|\mathrm{data}) = \frac{\bar{\pi}\mathrm{P}(data \mid noise)}{\bar{\pi}\mathrm{P}(data \mid noise)+(1-\bar{\pi})P(data \mid real)}$ reflects the probability that the subexon is 'noise', where 'data' is the observed information such as the coverage in each sample. We assume the samples are independent, hence $\mathrm{P}(\mathrm{data} \mid \mathrm{noise}) = \prod_{s=1}^{M} G_0^{(s)}(i)$, where s=1, ..., M denotes the sample. Here, $G_0^{(s)}(i)$ is $Gamma_{\theta_0^{(s)},k_0^{(s)}}(r_i^{(s)})$ if the *ratio model* is used for subexon i in sample s, and

$Gamma_{\theta'^{(s)}_0, k'^{(s)}_0}(c_i^{(s)})$ if the *coverage model* is employed. Similarly, $\mathrm{P}(\mathrm{data} \mid \mathrm{real}) = \prod_{s=1}^{M} G_1^{(s)}(i)$. In the end, the subexon is retained if it passes a pre-defined threshold.

### 2.2.4 Transcript selection

Candidate transcript models are represented as maximal paths in the global subexon graph, from a node with no incoming edges (source) to a node with no outgoing edges (sink). Since the graph generally encodes a much larger number of transcripts than is biologically possible, PsiCLASS identifies and selects a subset of transcripts that can explain all contiguity constraints from spliced reads and paired-reads. PsiCLASS first predicts a set of transcripts for each sample, using a graph-based dynamic programming algorithm with the global subexon graph and the sample specific alignment data, then combines the individual samples' transcript sets and selects a subset of meta-annotations by voting.

To predict a set of transcripts for each sample, PsiCLASS employs a SET_COVER framework and dynamic programming algorithms similar to its predecessor CLASS2 (Chapter 1 ), adapted for subexon graphs.

#### *SET_COVER formalism*

We define a constraint as a cluster of read alignments with the same subexon pattern. Like CLASS2, PsiCLASS uses constraints to decrease the memory usage while preserving the structural and contiguity information contained in the full set of reads. For a given graph G, let $C = \{c_1, \ldots, c_m\}$ denote the set of constraints and $T = \{t_1, \ldots, t_n\}$ the set of candidate transcripts, encoded in the graph. Given a constraint $c_i$, its abundance $a_i = a(c_i)$ defined as the number of supporting reads (or read pairs) normalized by the number of possible start positions of the

reads within the constraint's subexons. To reduce the transcript selection problem to SET_COVER, we view each candidate transcript $t_j$ as the set of constraints that are compatible with its exon-intron structure: $C(t_j) = \{c_1, \ldots, c_{nj}\}$, where $c_i \sim t_j$. In the simplest formulation, the goal then is to select a minimal (parsimonious) subset of transcripts that satisfies all constraints. More realistically, to account for the different abundance of constraints, we define a transcript's abundance as the minimum abundance among its set of constraints: $A_j = \min \{ a_i \mid c_i \sim t_j \}$. The goal then becomes to determine a subset of transcripts that most closely explain the constraints and their abundance levels. PsiCLASS uses a greedy approximation framework to address this problem, iteratively selecting the transcript that covers the largest number of constraints weighted by the constraints' abundance, then adjusting the constraints' abundance levels before the next iteration:

while ( {non-depleted constraints} $\neq \emptyset$ ):

(1) Choose transcript $t \in T$ that maximizes $|C(t)|(1 + \frac{A_t}{A})$

(2) Update the constraints' abundance:

$x = \min_{c \in C(t)} \{a(c)\}$

For each $c \in C(t)$:

$a(c) = a(c) - x$

if $a(c) \leq 0$, mark constraint *c* as depleted.

*Implementation:* PsiCLASS implements the procedure above in two steps. First, it determines the candidate set of transcripts *T*, using either enumeration (for graphs with <200,000 transcripts) or a variation of the splice-graph dynamic programming algorithm in Chapter 1 that

considers all reads single-end, for fast processing. Once the candidate transcript set *T* is determined, PsiCLASS applies the greedy SET_COVER approximation algorithm above.

For completeness, we include a brief description of the dynamic programming optimization procedure. The algorithm considers all subpaths *L*, and recursively calculates the maximum number of constraints *f(L)* for substranscripts starting with subpath *L*:

$$f(L) = \max_{L'} \{ f(L') + c(L,L'), \text{ if } L' \text{ exists; } c(L), \text{ if } L' \text{ does not exist } \},$$

where: *i)* L' is a subpath immediately following *L* so that all constraints compatible with *L* end before or within *L'*; *ii)* c(L,L') is the number of constraints starting in and (partially) compatible with L and L', and compatible with the concatenated subpath L.L'; and *iii)* c(L) is the number of constraints covered by subtranscript L. To take into account the abundance levels in the optimization process, at each sweep of the graph the algorithm excludes subpaths that cover constraints with abundance below a fixed value *x*; hence, the dynamic programming algorithm will return the best transcript with abundance greater than *x* (*x*-abundance transcript). With this modification, at each graph sweep the selection process selects an *x*-abundance transcript, starting with $x_0 = 0$ (thus guaranteeing that such a transcript exists), and each selected transcript's abundance value used as lower bound for the selection process at the following step: $0=x_0<x_1<x_2<\ldots<x_m$, until no transcript can be found. The optimal transcript then is among those selected by the sweeps. More details, along with proof of correctness for the algorithm, can be found in Chapter 1 .

### 2.2.5 Selecting a global set of transcripts

PsiCLASS selects a set of meta-annotations from the individual samples' sets of transcripts by voting. By default, at each locus a transcript is selected if it appears in a minimum number of samples, either pre-set or that can be specified by the user. As different voting parameter values might work best for data with specific characteristics, as exemplified by our liver RNA-seq collection, the user can adjust or re-calibrate the voting parameters post-assembly, starting from the full sets of transcripts of individual samples.

### 2.2.6 Performance evaluation scheme

#### *Sequence data*

We evaluated PsiCLASS and other methods on both simulated and real data. We generated 25 RNA-seq samples, with ~85 million 100 bp paired-end reads, using the software Polyester [48] with the default gene and transcript distribution models and randomly sampling 10% of the transcripts (at 13,912 genes) from the human GENCODE v.27 [22] gene annotations. Reads were aligned to the reference genome hg38 separately with HISAT2 [1] and STAR [2]. Chromosome 2 alignments were extracted and used in the assembly and evaluations. Human liver RNA-seq samples were obtained from the Stanley Foundation and previously sequenced by Dr. Sabunciyan's lab. Total RNA was isolated using the Qiagen RNeasy kit and libraries were constructed using the Illumina TruSeq Stranded Total RNA kit for Human/Mouse/Rat following the manufacturers recommended protocol. The resulting stranded, rRNA depleted liver libraries were sequenced on an Illumina HiSeq 2000 instrument.  667 RNA-seq samples from human lymphoblastoid cell lines part of the GEUVADIS population variation project were publicly

available from ArrayExpress (accession: E-GEUV-6), and mouse hippocampus RNA-seq data were those reported in [49] and available from GenBank (ProjectID: PREJB18790).

#### *Evaluation metrics*

Once the reads were mapped to the genome, we used StringTie v.1.3.3.b and Scallop v.0.10.2 to assemble them into transcripts, for each individual sample. Transcript sets for all samples in an experiment were then merged with StringTie(ST)-merge and TACO v.0.7.3. For PsiCLASS v.1.0.0, reads were assembled simultaneously across all samples. To evaluate the accuracy of transcript assembly, we employed standard sensitivity (Sn) and precision (Pr) measures and evaluation criteria to assess the accuracy of transcript reconstructions by comparison to a gold reference. The reference annotation for the simulation experiment consisted of the set of transcripts simulated by Polyster, whereas experiments on real data used the human GENCODE v.27 and mouse RefSeq gene annotations. A predicted transcript is deemed a true positive (TP) iff its intron chain fully matches that of a gold reference transcript. If N is the number of predicted transcripts, M be the number of ground truth transcripts, then Sn = TP/M and Pr = TP/N [11].

## 2.3 Results

### 2.3.1 Performance on simulated data

We compared PsiCLASS with the best current approaches, namely StringTie and Scallop at the single-sample level, and the combinations of StringTie with ST-merge and Scallop with TACO, at the meta-assembly level. (Other combinations are shown in Figure 2-2). We first applied the methods to 25 RNA-seq samples simulated with Polyester [48], where reads were aligned with

two methods, Hisat2 [1] and STAR [2]. Performance was slightly better for all programs when reads were aligned with Hisat2 (Figure 2-3**)**, therefore we chose this alignment method for the rest of the analyses.

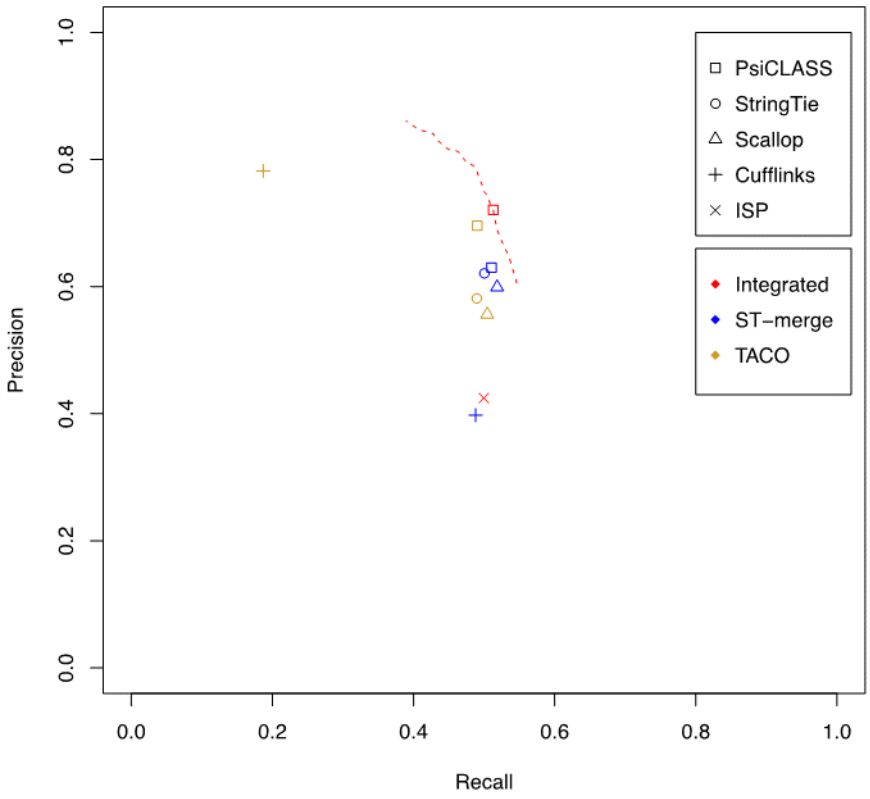



*Figure 2-2 Performance evaluation of combination methods at the level of meta-annotations on simulated data*

*(Methods tested include three single-sample assemblers (Cufflinks, StringTie and Scallop), two meta-assemblers (TACO and StringTie(ST)-merge), and two multi-sample integrated methods (PsiCLASS and ISP), where TACO and ST-merge were used to aggregate the outputs from individual samples into a unified set of meta-annotations. Below, the shape of the point represents the tool used, and the color represents the aggregation method. For PsiCLASS, the red curve shows the variation in performance as the voting cutoff varies between 1 to 25 (right to left). PsiCLASS produces the highest precision regardless of the meta-assembly method used (TACO, ST-merge or the PsiCLASS3 built-in voting), and its sensitivity is comparable with the best of the other methods.)*

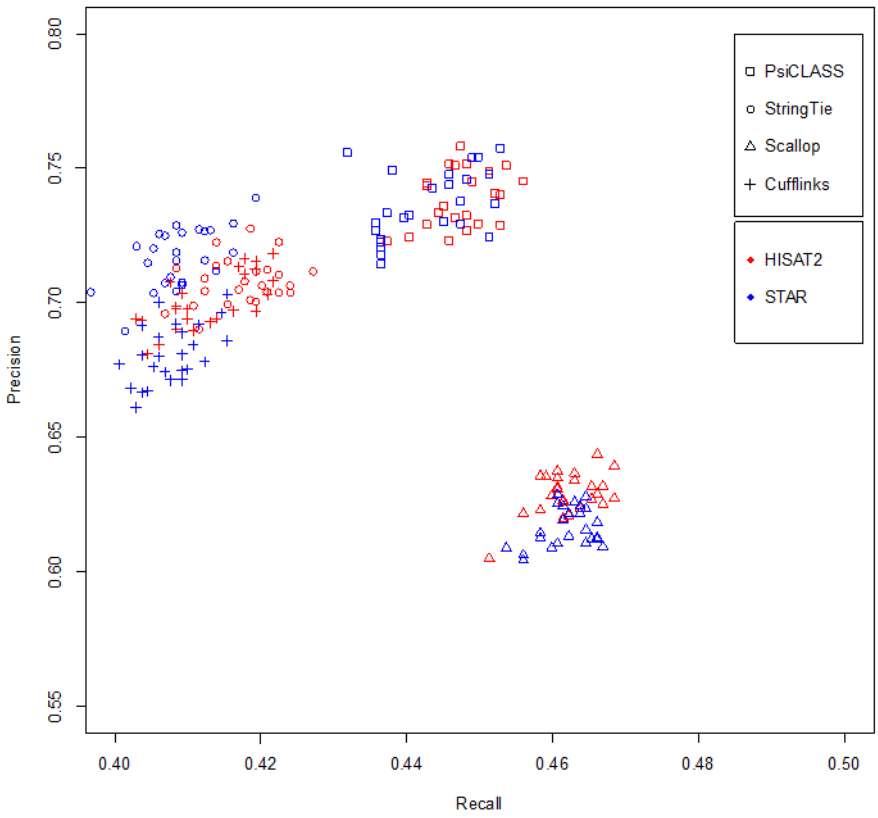



*Figure 2-3 Comparison of transcript assembly methods at the single sample-level, and with different alignment tools*

*(Each point represents the performance of the stated method on one of the 25 simulated samples. The shape of the mark represents the transcript assembly method (StringTie, Scallop and PsiCLASS), and the color indicates the RNA-seq alignment tool (Hisat2 and STAR). All methods perform similarly with the two alignment methods, with Hisat2 leading to a slight increase in performance. When assembly methods are compared, PsiCLASS using a global subexon graph leads to improved accuracy at single-sample level, with the highest per sample average precision, and sensitivity slightly higher than StringTie's (by 3%) and comparable to Scallop's (within 1.5%).)*

On the simulated data, PsiCLASS with default voting achieved 72.1% precision, which is 16.1% higher than the StringTie system and 29.5% higher than Scallop with TACO, whereas sensitivity for all programs was roughly 50% (Figure 2-4). Even at the individual sample level, PsiCLASS had both the highest precision and the highest sensitivity: 75.8% precision on average, compared to 70.8% for StringTie and 62.9% for Scallop, and 47.8% sensitivity compared to 41.7% for

StringTie and 46.2% for Scallop. Precision values for both StringTie and Scallop, but to a lesser extent for PsiCLASS, dropped significantly after aggregation, hence PsiCLASS produces more consistent sets of transcripts between individual samples and the set of meta-annotations.

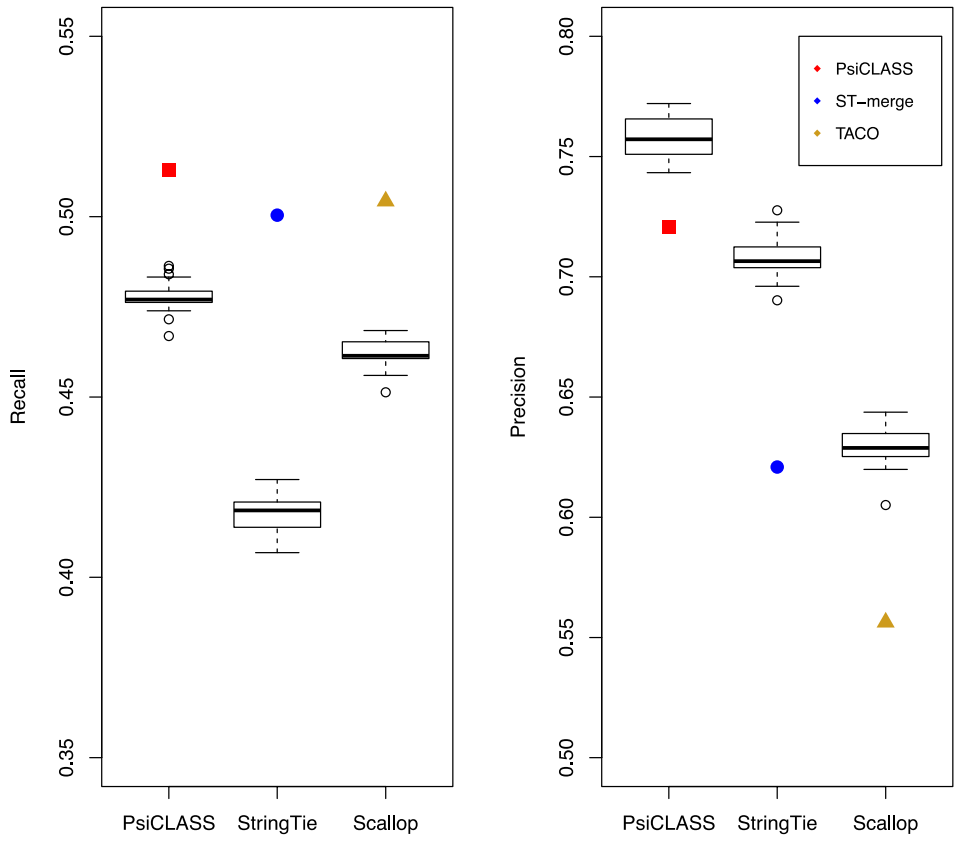



*Figure 2-4 Performance evaluation of methods on 25 simulated data sets*

*(Sensitivity (recall) and precision values for PsiCLASS, StringTie and Scallop at the level of individual samples are shown in boxed plots, and meta-annotations resulted from aggregation (with PsiCLASS voting, ST-merge and TACO) are shown with colored shapes.)*

### *Performance by transcript expression level*

We further investigated the performance of methods based on the transcripts' expression levels (Figure 2-5). Simulated transcripts were divided into low (463 transcripts; Fragments Per

Kilobase (FPK)<30), medium (658 transcripts; 30<=FPK<500) and high (322 transcripts; FPK>=500) according to the pre-defined expression levels. PsiCLASS with voting reconstructs the largest fraction of highly-expressed transcripts, 82.4%, and all three programs recover ~60% of the medium–expressed ones. Scallop shows good sensitivity in detecting low expression features, confirming prior reports [46], followed closely by StringTie. Overall, however, the sensitivity of all programs on this class of transcripts is very low, between 10% and 20%. Note that, because a reconstructed transcript's expression level may fall in another class than the predefined one, precision cannot be rigorously evaluated.

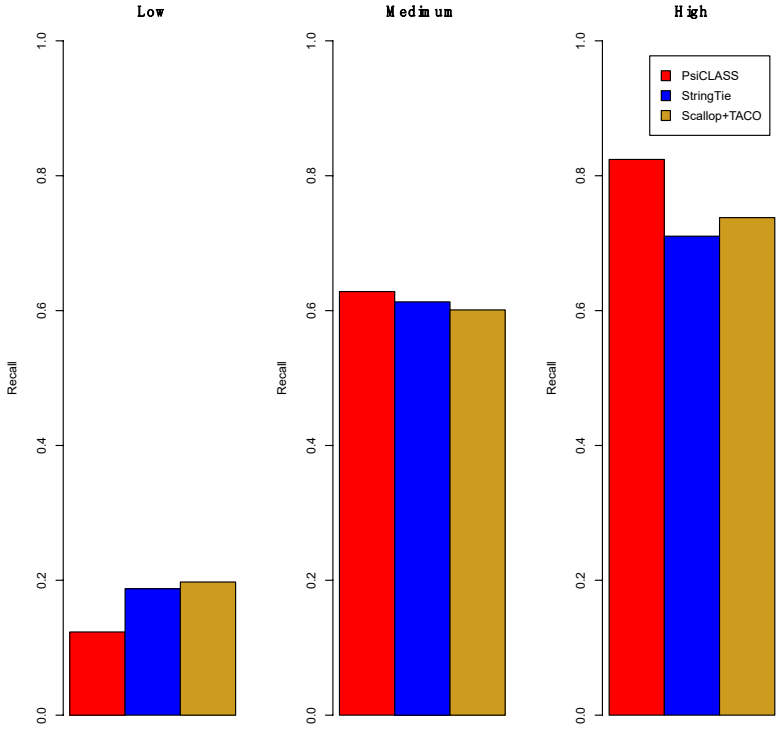



*Figure 2-5 Performance evaluation of methods on 25 simulated sets, genes grouped by abundance*

### 2.3.2 Performance on real RNA-seq data

We next assessed the performance on two representative RNA-seq data sets, generated with two different library preparation protocols: 25 randomly selected sets from polyA-selected lymphoblastoid samples from the GEUVADIS population variation project, and 73 rRNA-depleted total RNA libraries from postmortem human liver samples with funding from the Stanley Medical Research Institute. With the default voting setting, for the GEUVADIS set PsiCLASS' precision is 83% and 104% higher than StringTie's and Scallop plus TACO's, respectively, whereas sensitivity is 2.4% and 15.2% lower (Figure 2-6). Notably, precision remains higher as the voting cutoff varies (Figure 2-8A), exceeding StringTie's by 83% and Scallop's by 66% even as PsiCLASS matches or approaches each method's sensitivity setting. Similarly, advantages of the multi-sample approach are seen for the liver total RNA data set, with 257% and 331% improvements in precision, albeit at 19% and 36% lower sensitivity. In such cases, the default voting setting may not present the best tradeoff, and as PsiCLASS' precision remains significantly higher than that of its counterparts the user may choose a different cutoff (Figure 2-8B). For instance, when sensitivity is matched to that of StringTie and Scallop, PsiCLASS maintains a 149% and 45% increase in precision over these systems, respectively (Figure 2-7).

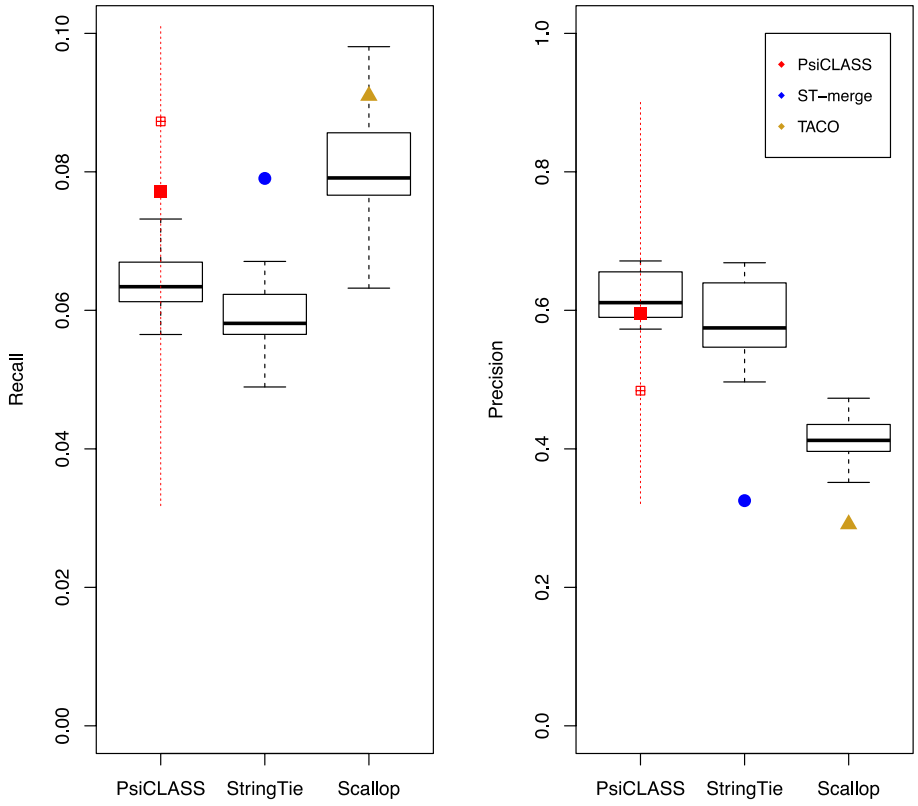



*Figure 2-6 Performance evaluation of methods on 25 GEUVADIS samples (poly-adenylated RNA)*

*(Additional symbols mark sensitivity and precision values for PsiCLASS when tuned to match or approach the sensitivity of its competitor)*

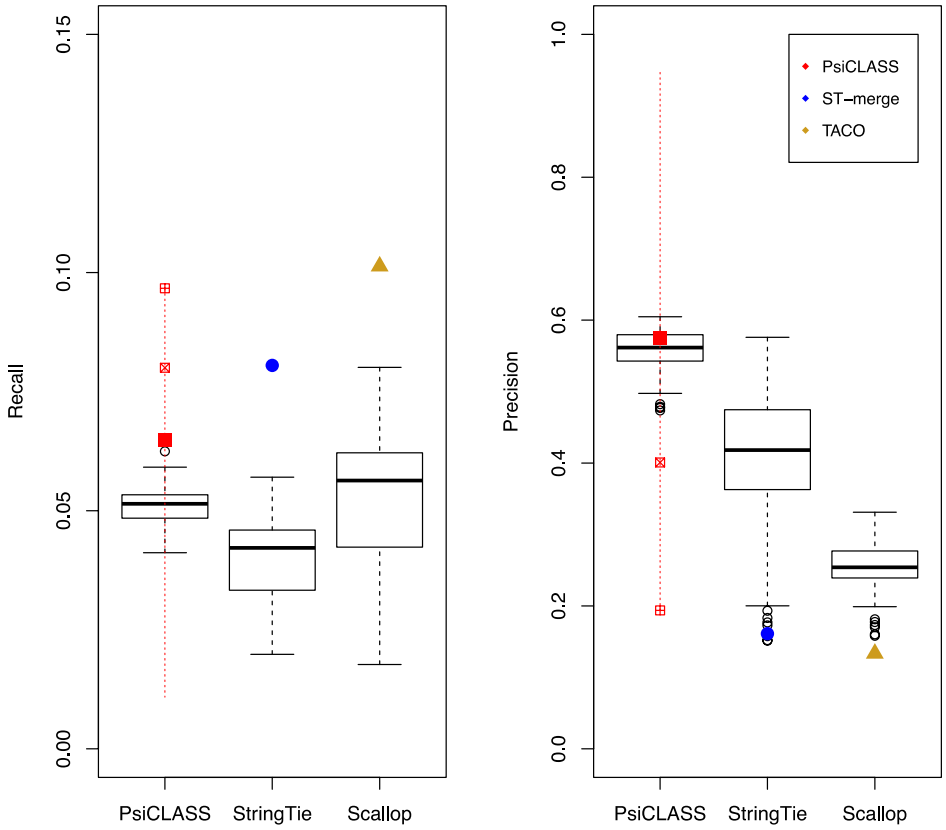



*Figure 2-7 Performance evaluation of methods on 73 liver RNA-seq samples (rRNA-depleted total RNA)*

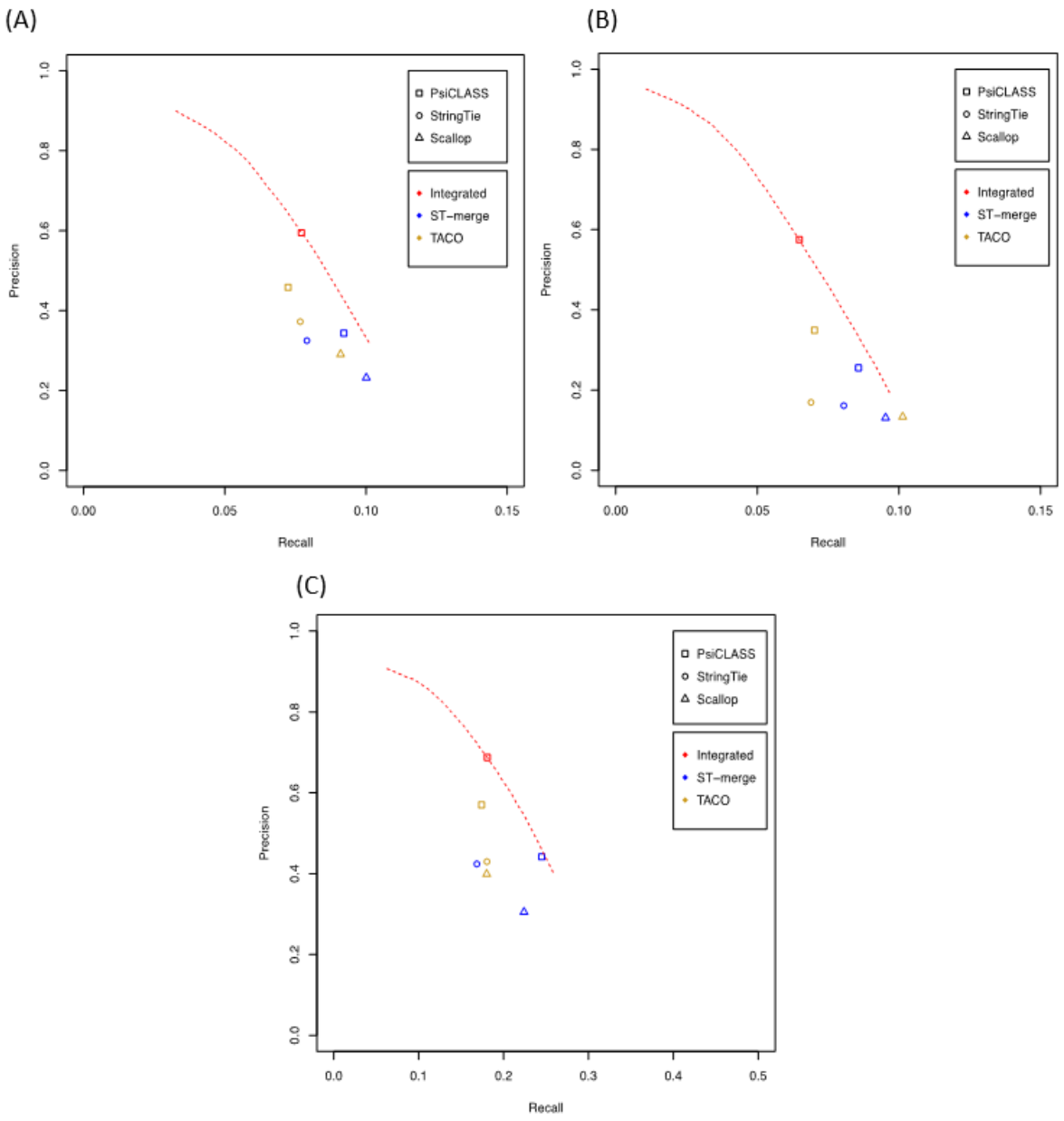



*Figure 2-8 Performance evaluation of methods on real data*

*((A) all method combinations, meta-annotations, Geuvadis data (25 samples); (B) all method combinations, meta-annotations, total RNA from human liver (73 samples); and (C) all method combinations, meta-annotations, mouse hippocampus samples, healthy and with induced epileptic seizures (44 samples). In (A-C), the voting cutoff for the minimum number of samples that a PsiCLASS3-reported transcript must appear in is varied between 1 and 25 (Geuvadis), 1 and 73 (human liver), and 1-44 (mouse hippocampi), respectively (shown left-to-right, as red curves).)*

### *Performance on two-condition data*

Most RNA-seq analyses are aimed at determining differences between conditions. To explore the robustness of PsiCLASS when combining multi-condition samples, we applied it and the other methods to RNA-seq samples from hippocampi of normal mice (24 samples) and mice with induced epileptic seizures (20 samples) [49]. The diagrams in Figure 2-9 indicate that at the level of individual samples PsiCLASS has slightly higher sensitivity than StringTie, by 6% on average, but mildly lower than Scallop, by 9%. However, after voting, PsiCLASS' precision at the level of meta-annotations is 62.1% and 72.4% higher than the other programs', along with a slight increase in sensitivity, therefore recommending it as the overall best performer. Thus, PsiCLASS can be effectively and more reliably used on the aggregate set of samples in a two-condition comparison.

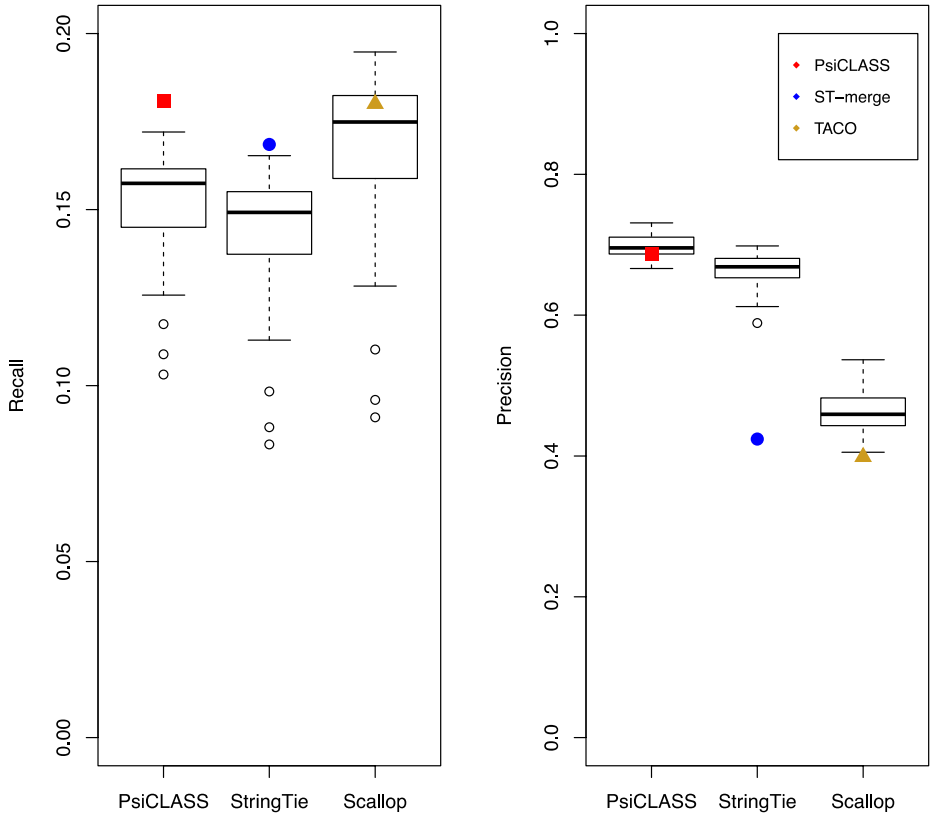



*Figure 2-9 Performance evaluation of methods on 44 hippocampus samples from healthy and epileptic mice*

### 2.3.3 Scalability

We next investigated the utility of the multi-sample approach for small data collections, including single samples. Table 2-1 shows the results for all methods on sets of 2, 3, 5 and 10 samples from our simulated set, averaged over five independent trials, and for single-sample sets, averaged over all possible 25 trials. The multi-sample approach has clear benefits in all cases, most notably for precision (71-81% for PsiCLASS, compared to 64-71% for StringTie, and 57-63% for Scallop with TACO), at comparable levels of sensitivity (42-50% for all three

methods). Remarkably, PsiCLASS ranks slightly ahead of both StringTie and Scallop as single-sample assemblers, and therefore can be gainfully used even within the conventional approach.

*Table 2-1 Performance of methods on experiments with small numbers of samples*

| | Recall (%) | | | Precision (%) | | |
|---|---|---|---|---|---|---|
| Sample Size | PsiCLASS | StringTie | Scallop | PsiCLASS | StringTie | Scallop |
| 1 | 44.7 | 41.7 | 46.2 | 73.8 | 70.8 | 62.9 |
| 2 | 42.5 | 43.8 | 43.3 | 78.5 | 69.1 | 57.0 |
| 3 | 42.3 | 45.2 | 45.0 | 81.1 | 68.5 | 56.7 |
| 5 | 47.0 | 46.6 | 46.2 | 77.6 | 66.7 | 56.7 |
| 10 | 50.4 | 48.0 | 48.1 | 71.6 | 64.0 | 56.5 |

As the emerging landscape of RNA sequencing foresees increasingly larger data sets from large patient cohorts and population variation studies, we aimed to assess the suitability of the multi-sample approach as the data set increases. We evaluated program performance on increasingly larger subsets of RNA-seq samples from the GEUVADIS population variation project, up to the full set of 667 samples (Figure 2-10). All methods show improvements in sensitivity as the number of samples increases, but while PsiCLASS and Scallop show further slight gains after 20-50 samples, the sensitivity of StringTie drops. Precision drops markedly for both Scallop and StringTie, to less than 35% for 50 samples and below 20% for the full set of samples. In sharp contrast, PsiCLASS's sensitivity and precision remain almost constant with more than 10 samples, demonstrating the robustness of this approach. Also, with sensitivity comparable to StringTie's and precision (75%) twice as high as that of the other two systems when the data set exceeds 20-50 samples, PsiCLASS is unequivocally the best suited for handling large RNA-seq collections. Lastly, PsiCLASS took only 9 hours with 24 threads to process the 667 samples on an 3.0 GHz Intel "Ivy Bridge" Xeon server, amounting to less than 1 minute per sample.

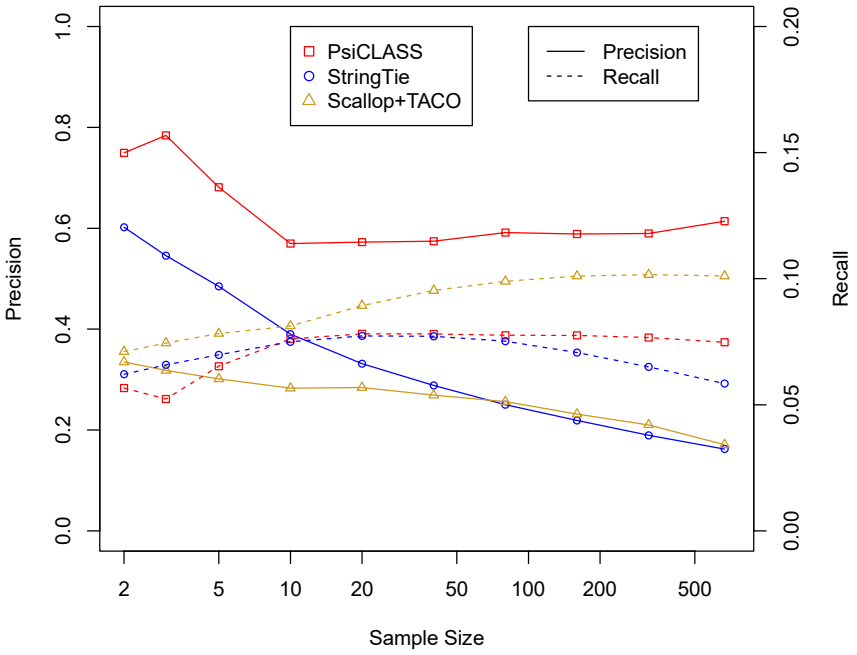



*Figure 2-10 Performance evaluation of methods on 667 GEUVADIS samples*

## 2.4 Conclusions

Determining the set of expressed genes and transcripts in an RNA-seq experiment is critical for subsequent quantification and differential expression and splicing analyses. The conventional approach to process each sample separately and then merge the sets of transcripts to create a unified set of annotations has limitations, in particular low precision. We present PsiCLASS, a transcript assembler and meta-assembler that simultaneously analyzes all samples in an RNA-seq experiment. It employs global subexon graphs along with statistical models of intronic read coverage, dynamic programming optimization algorithms, and voting for transcript selection. PsiCLASS has significantly higher precision at similar sensitivity when compared to current best methods; precision remains consistently high, over 55%, when tested on data from a variety of experimental conditions. PsiCLASS is scalable, efficient and robust with large numbers of RNA-

seq samples, thus providing a highly effective paradigm for large-scale analyses of collections of hundreds and thousands of samples.

Development and evaluations were performed on the Maryland Advanced Research Computing Center (MARCC). Work was supported in part by NSF grants ABI-1356078 and IOS-1339134 to L.F., and by NIH grant R01GM12453 to L.F. and Kathleen Burns. S.S was supported by a grant from the Stanley Medical Research Institute. PsiCLASS is available free of charge from https://github.com/splicebox/PsiCLASS .

# Chapter 3

# Lighter: fast and memory-efficient error correction without counting

## 3.1 Introduction

The cost and throughput of DNA sequencing have improved rapidly in the past several years [50], with recent advances reducing the cost of sequencing a single human genome at 30-fold coverage to around $1,000 [51]. With these advances has come an explosion of new software for analyzing large sequencing datasets. Sequencing error correction is a basic need for many of these tools. Removing errors can also improve the accuracy, speed and memory-efficiency of downstream tools, particularly for *de novo* assemblers based on De Bruijn graphs [52, 53].

To be useful in practice, error correction software must make economical use of time and memory even when input datasets are large (many billions of reads) and when the genome under study is also large (billions of nucleotides). Several methods have been proposed, covering a wide tradeoff space between accuracy, speed and memory- and storage-efficiency. SHREC [54] and HiTEC [55] build a suffix index of the input reads and locate errors by finding instances where a substring is followed by a character less often than expected. Coral [56] and ECHO [57] find overlaps among reads and use the resulting multiple alignments to detect and

correct errors. Reptile [58] and Hammer [59] detect and correct errors by examining each k-mer's neighborhood in the dataset's k-mer Hamming graph.

The most practical and widely used error correction methods descend from the spectral alignment approach introduced in the earliest De Bruijn graph based assemblers [52, 53]. These methods count the number of times each k-mer occurs (its multiplicity) in the input reads, then apply a threshold such that k-mers with multiplicity exceeding the threshold are considered solid. These k-mers are unlikely to have been altered by sequencing errors. k-mers with low multiplicity (weak k-mers) are systematically edited into high-multiplicity k-mers using a dynamic-programming solution to the spectral alignment problem [52, 53] or, more often, a fast heuristic approximation. Quake [60], one of the most widely used error correction tools, uses a hash-based k-mer counter called Jellyfish [61] to determine which k-mers are correct. CUDA-EC [62] was the first to use a Bloom filter as a space-efficient alternative to hash tables for counting k-mers and for representing the set of solid k-mers. More recent tools, such as Musket [63] and BLESS [64], use a combination of Bloom filters and hash tables to count k-mers or to represent the set of solid k-mers.

Lighter (LIGHTweight ERror corrector) is also in the family of spectral alignment methods, but differs from previous approaches in that it avoids counting k-mers. Rather than count k-mers, Lighter samples k-mers randomly, storing the sample in a Bloom filter. Lighter then uses a simple test applied to each position of each read to compile a set of solid k-mers, stored in a second Bloom filter. These two Bloom filters are the only sizable data structures used by Lighter.

A crucial advantage is that Lighter's parameters can be set such that memory footprint and accuracy are near constant with respect to depth of sequencing. That is, no matter how deep the coverage, Lighter can allocate the same sized Bloom filters and achieve nearly the same: (a) Bloom filter occupancy, (b) Bloom filter false positive rate and (c) error correction accuracy. Lighter does this without using any disk space or other secondary memory. This is in contrast to BLESS and Quake/Jellyfish, which use secondary memory to store some or all of the k-mer counts.

Lighter's accuracy is comparable to competing tools. We show this both in simulation experiments where false positives and false negatives can be measured, and in real-world experiments where read alignment scores and assembly statistics can be measured. Lighter is also very simple and fast, faster than all other tools tried in our experiments. These advantages make Lighter quite practical compared to previous counting-based approaches, all of which require an amount of memory or secondary storage that increases with depth of coverage. Lighter is free open-source software available from https://github.com/mourisl/Lighter.

## 3.2 Methods

Lighter's workflow is illustrated in Figure 3-1. Lighter makes three passes over the input reads. The first pass obtains a sample of the *k*-mers present in the input reads, storing the sample in Bloom filter A. The second pass uses Bloom filter A to identify solid *k*-mers, which it stores in Bloom filter B. The third pass uses Bloom filter B and a greedy procedure to correct errors in the input reads.

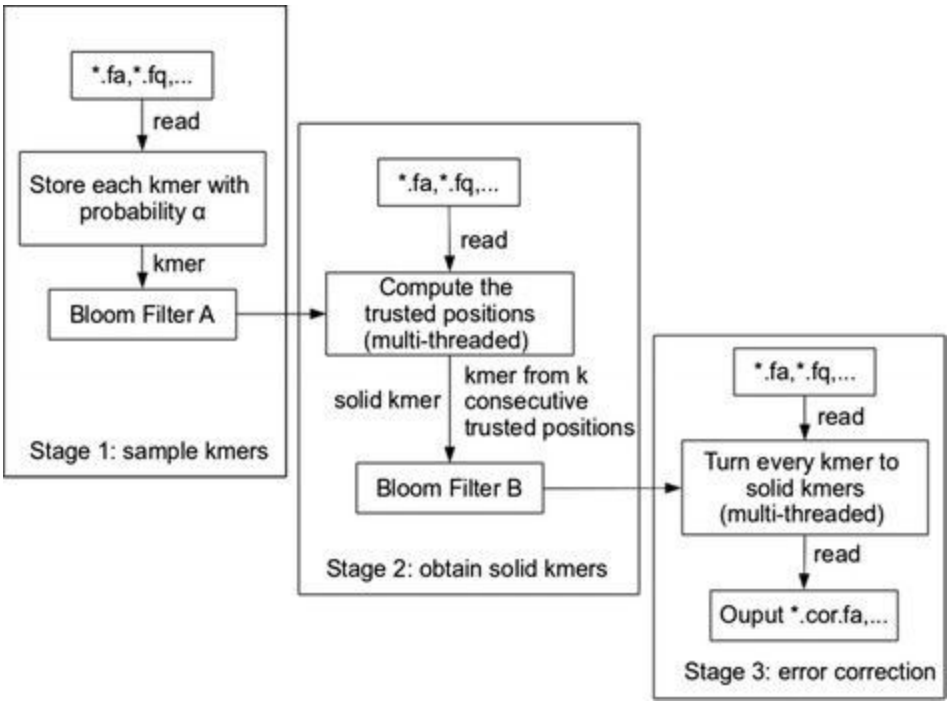



*Figure 3-1 The framework of Lighter*

### 3.2.1 Bloom filter

A Bloom filter [65] is a compact probabilistic data structure representing a set. It consists of an array of m bits, each initialized to 0. To add an item o, h independent hash functions $H_0(o),H_1(o),\ldots,H_{h-1}(o)$ are calculated. Each maps o to an integer in [0, m) and the corresponding h array bits are set to 1. To test if item q is a member, the same hash functions are applied to q. q is a member if all corresponding bits are set to 1. A false positive occurs when the corresponding bits are set to 1 'by coincidence', that is, because of items besides q that were added previously. Assuming the hash functions map items to bit array elements with equal probability, the Bloom filter's false positive rate is approximately $\left(1-e^{-h\frac{n}{m}}\right)^h$, where n is the number of distinct items added, which we call the cardinality. Given n, which is usually determined by the dataset, m and h can be adjusted to achieve a desired false positive rate.

Lower false positive rates can come at a cost, since greater values of m require more memory and greater values of h require more hash function calculations. Many variations on Bloom filters have been proposed that additionally permit compression of the filter, storage of count data, representation of maps in addition to sets, etc. [66]. Bloom filters and variants thereon have been applied in various bioinformatics settings, including assembly [67][19], compression [68], k-mer counting [69] and error correction [62].

By way of contrast, another way to represent a set is with a hash table. Hash tables do not yield false positives, but Bloom filters are far smaller. Whereas a Bloom filter is an array of bits, a hash table is an array of buckets, each large enough to store a pointer, key or both. If chaining is used, lists associated with buckets incur additional overhead. While the Bloom filter's small size comes at the expense of false positives, these can be tolerated in many settings including in error correction.

Lighter's efficiency depends on the efficiency of the Bloom filter implementation. For a standard Bloom filter, each of the h hash functions could map item o to any element of the bit array. The bit array will often be very large, much larger than the processor cache. Thus, each probe into the bit array is likely to cause a cache miss. Putze et al [70] propose a blocked Bloom filter. Given a block size b, the first hash function $H_0(o)$ is used to select a size-b block of consecutive positions in the bit array. Then, $H_1(o),...,H_{h-1}(o)$ map o onto elements of that block. When b is less than or equal to the size of a cache line, the h accesses will tend to cause only one or two cache misses, rather than approximately h cache misses. The drawback is that h and m must be somewhat larger to achieve the same false positive rate (FPR) as a corresponding standard Bloom filter. To estimate the FPR of the blocked Bloom filter, we can consider each of

the possible m − b + 1 blocks. For the i-th block, the FPR within the block is $(b'_i/b)^h$, where $b'_i$ is the number of bits set to 1 in block i. So the overall FPR is:

$$\frac{\sum_i \left(\frac{b'_i}{b}\right)^h}{m - b + 1}$$

Putze et al also propose a pattern-blocked Bloom filter [70], where the difference is that instead of updating the h positions in the block separately, we pre-compute a list of patterns where each pattern is a bitmask describing how to update h positions in a block with a few bitwise operations. To perform such an update we first find the appropriate pattern using hash function, then update the corresponding positions simultaneously. In Lighter, 64-bit integers are used to form the mask. For example, if b = 256, the pattern is made up of 4 64-bit integers, and we can update in 4 64-bit operations, regardless of h. The FPR formula above still roughly estimates the FPR for the pattern-blocked bloom filter.

In our method, the items to be stored in the Bloom filters are k-mers. Because we would like to treat genome strands equivalently for counting purposes, we will always canonicalize a k-mer before adding it to or using it to query a Bloom filter. A canonicalized k-mer is either the k-mer itself or its reverse complement, whichever is lexicographically prior.

### 3.2.2 Sequencing model

We use a simple model to describe the sequencing process and Lighter's subsampling. The model resembles one suggested previously [71]. Let K be the total number of k-mers obtained by the sequencer. We say a k-mer is incorrect if its sequence has been altered by one or more sequencing errors. Otherwise it is correct. Let ε be the fraction of k-mers that are incorrect. We

assume ε does not vary with the depth of sequencing. The sequencer obtains correct k-mers by sampling independently and uniformly from k-mers in the genome. Let the number of k-mers in the genome be G, and assume all are distinct. If κ c is a random variable for the multiplicity of a correct k-mer in the input, $\kappa_c$ is binomial with success probability 1/G and number of trials (1−ε)K:

$$\kappa_c \sim \text{Binom}((1-\varepsilon)K, 1/G).$$

Since the number of trials is large and the success probability is small, the binomial is well approximated by a Poisson:

$$\kappa_c \sim \text{Pois}(K(1-\varepsilon)/G).$$

A sequenced k-mer survives subsampling with probability $\alpha$. If $\kappa'_c$ is a random variable for the number of times a correct k-mer appears in the subsample:

$$\kappa'_c \sim \text{Binom}((1-\varepsilon)K, \alpha/G),$$

which is approximately Pois(αK(1−ε)/G).

We model incorrect k-mers similarly. The sequencer obtains incorrect k-mers by sampling independently and uniformly from k-mers 'close to' a k-mer in the genome. We might define these as the set of all k-mers with low but non-zero Hamming distance from some genomic k-mer. If $\kappa_e$ is a random variable for the multiplicity of an incorrect k-mer, $\kappa_e$ is binomial with success probability 1/H and number of trials εK: $\kappa_e \sim \text{Binom}(\varepsilon K, 1/H)$, which is approximately Pois(Kε/H). It is safe to assume $H \gg G$. $\kappa'_e \sim \text{Pois}(\alpha K \varepsilon / H)$ is a random variable for the number of times an incorrect k-mer appears in the subsample.

Others have noted that, given a dataset with deep and uniform coverage, incorrect k-mers occur rarely while correct k-mers occur many times, proportionally to coverage [52, 53].

### 3.2.3 Stages of the method

#### *First pass*

In the first pass, Lighter examines each k-mer of each read. With probability 1−α, the k-mer is ignored. k-mers containing ambiguous nucleotides (e.g. 'N') are also ignored. Otherwise, the k-mer is canonicalized and added to Bloom filter A.

Say a distinct k-mer a occurs a total of $N_a$ times in the dataset. If none of the $N_a$ occurrences survive subsampling, the k-mer is never added to A and A's cardinality is reduced by one. Thus, reducing α can in turn reduce A's cardinality. Because correct k-mers are more numerous, incorrect k-mers tend to be discarded from A before correct k-mers as α decreases.

The subsampling fraction α is set by the user. We suggest adjusting α in inverse proportion to depth of sequencing, for reasons discussed below. For experiments described here, we set α=0.1 when the average coverage is 70-fold. That is, we set α to 0.1(70/C), where C is average coverage.

#### *Second pass*

A read position is overlapped by up to x k-mers, $1 \leqslant x \leqslant k$, where x depends on how close the position is to either end of the read. For a position altered by sequencing error, the overlapping k-mers are all incorrect and are unlikely to appear in A. We apply a threshold such that if the number of k-mers overlapping the position and appearing in Bloom filter A is less than the threshold, we say the position is untrusted. Otherwise we say it is trusted. Each instance where

the threshold is applied is called a test case. When one or more of the x k-mers involved in two test cases differ, we say the test cases are distinct.

Let $P^*(\alpha)$ be the probability an incorrect k-mer appears in A, taking the Bloom filter's false positive rate into account. If random variable $B_{e,x}$ represents the number of k-mers appearing in A for an untrusted position overlapped by x k-mers:

$$B_{e,x} \sim \text{Binom}(x, P^*(\alpha)).$$

We define thresholds $y_x$, for each x in [1,k]. $y_x$ is the minimum integer such that:

$$P(B_{e,x} \leqslant y_x - 1) \geqslant 0.995.$$

Ignoring false positives for now, we model the probability of a sequenced k-mer having been added to A as:

$$P(\alpha) = 1 - (1 - \alpha)^{f(\alpha)}.$$

We define:

$$f(\alpha) = \max\{2, 0.2/\alpha\}.$$

That is, we assume the multiplicity of a weak k-mer is at most f(α), which will often be a conservative assumption, especially for small α. It is also possible to define P(α) in terms of random variables $K_e$ and $K_e'$, but we avoid this here for simplicity.

A property of this threshold is that when α is small:

$$P(\alpha/z) = 1 - \left(1 - \frac{\alpha}{z}\right)^{0.2z/\alpha} \approx 1 - (1 - \alpha)0.2/\alpha = P(\alpha),$$

where z is a constant greater than 1 and we use the fact that:

$$(1-\alpha/z)^z \approx 1-\alpha.$$

For $P^*(\alpha)$, we additionally take A's false positive rate into account. If the false positive rate is β, then:

$$P^*(\alpha)=P(\alpha)+\beta-\beta P(\alpha).$$

Once all positions in a read have been marked trusted or untrusted using the threshold, we find all instances where k trusted positions appear consecutively. The k-mer made up by those positions is added to Bloom filter B.

### *Third pass*

In the third pass, Lighter applies a simple, greedy error correction procedure like that used in BLESS [64]. A read r of length $|r|$, contains $|r|-k+1$ k-mers. $k_i$ denotes the k-mer starting at read position i, $1 \leqslant i \leqslant |r|-k+1$. We first identify the longest stretch of consecutive k-mers in the read that appear in Bloom filter B. Let $k_b$ and $k_e$ be the k-mers at the left and right extremes of the stretch. If $e<|r|-k+1$, we examine successive k-mers to the right starting at $k_e+1$. For a k-mer $k_i$ that does not appear in B, we assume the nucleotide at offset $i+k-1$ is incorrect. We consider all possible ways of substituting for the incorrect nucleotide. For each substitution, we count how many consecutive k-mers starting with $k_i$ appear in Bloom filter B after making the substitution. We pick the substitution that creates the longest stretch of consecutive k-mers in B. The procedure is illustrated in Figure 3-2.

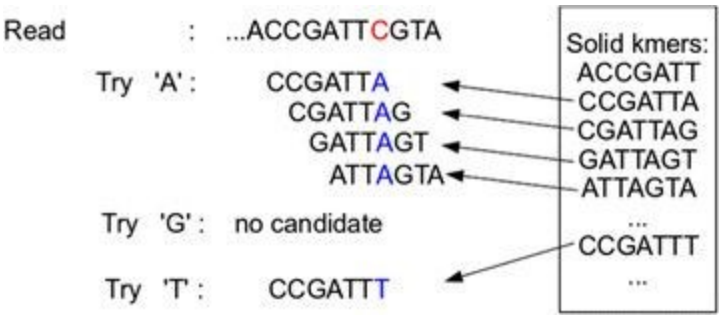



*Figure 3-2 An example of the greedy error correction procedure. k-mer CCGATTC does not appear in Bloom filter B, so we attempt to substitute a different nucleotide for the C shown in red. We select A since it yields the longest stretch of consecutive k-mers that appear in Bloom filter B.*

If more than one candidate substitution is equally good (i.e. results in the same number of consecutive k-mers from B), we call position i+k−1 ambiguous and make no attempt to correct it. The procedure then resumes starting at $k_{i+k}$, or the procedure ends if the read is too short to contain k-mer $k_{i+k}$.

When errors are located near to the end of a read, the stretches of consecutive k-mers used to prioritize substitutions are short. For example, if the error is at the very last position of the read, we must choose a substation on the basis of just one k-mer: the rightmost k-mer. This very often results in a tie, and no correction. Lighter avoids many of these ties by considering k-mers that extend beyond the end of the read. Lighter extends the read base by base. For the new base beyond the read, Lighter tries all the four nucleotides in the order of “A”, “C”, “G”, “T”, and uses the first nucleotide creating a k-mer that can be found in Bloom filter A. This procedure is terminated until all the nucleotides fails or the distance to the candidate substitution’s position is larger than k-1. Then we choose the candidate substitution with the longest extension based on this greedy procedure. As a result, we can solve some ties that are more likely to happened near the end of a read due to insufficient extension.

For better precision, Lighter also limits the corrections that can be made in any window of size k in a read. The default limit is 4, and it is configurable. Corrections at positions with an 'N' contribute 0, and corrections at low-quality bases (defined in the Quality score section below) contribute 0.5 toward this limit. All other positions contribute 1.

### 3.2.4 Scaling with depth of sequencing

Lighter's accuracy can be made near constant as the depth of sequencing K increases and its memory footprint is held constant. This is accomplished by holding αK constant, i.e., by adjusting α in inverse proportion to K. This is illustrated in Table 3-1 and (Rows labeled k show the k-mer sizes selected for each tool and dataset)

Table 3-2.

*Table 3-1 Accuracy measures for datasets simulated with Mason with various sequencing depths and error rates*

| Coverage | | 35× | | 70× | | 140× | |
|---|---|---|---|---|---|---|---|
| Error rate | | 1% | 3% | 1% | 3% | 1% | 3% |
| *α* for Lighter | | 0.2 | 0.2 | 0.1 | 0.1 | 0.05 | 0.05 |
| Recall | Quake | 89.68 | 48.77 | 89.64 | 48.82 | 89.59 | 48.78 |
| | SOAPec | 57.71 | 38 | 57.57 | 37.71 | 57.09 | 36.76 |
| | Musket | 93.75 | 92.62 | 93.73 | 92.64 | 93.73 | 92.63 |
| | Bless | 99.81 | 99.33 | 99.82 | 99.58 | 99.82 | 99.58 |
| | Lighter | 99.87 | 98.53 | 99.84 | 98.72 | 99.86 | 98.78 |
| Precision | Quake | 99.99 | 99.99 | 99.99 | 99.99 | 99.99 | 99.99 |
| | SOAPec | 99.99 | 100 | 99.99 | 99.99 | 99.99 | 99.99 |
| | Musket | 99.99 | 99.93 | 99.99 | 99.93 | 99.99 | 99.93 |
| | Bless | 99.73 | 98.86 | 99.73 | 99.35 | 99.72 | 99.36 |
| | Lighter | 99.98 | 99.96 | 99.98 | 99.96 | 99.98 | 99.96 |
| F-score | Quake | 94.55 | 65.56 | 94.54 | 65.61 | 94.51 | 65.57 |
| | SOAPec | 73.18 | 55.07 | 73.07 | 54.77 | 72.68 | 53.75 |
| | Musket | 96.77 | 96.14 | 96.76 | 96.15 | 96.76 | 96.15 |
| | Bless | 99.77 | 99.09 | 99.77 | 99.47 | 99.77 | 99.47 |
| | Lighter | 99.93 | 99.24 | 99.91 | 99.33 | 99.92 | 99.36 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Gain | Quake | 89.67 | 48.76 | 89.64 | 48.82 | 89.59 | 48.78 |
| | SOAPec | 57.7 | 38 | 57.57 | 37.71 | 57.09 | 36.75 |
| | Musket | 93.74 | 92.56 | 93.72 | 92.58 | 93.72 | 92.57 |
| | Bless | 99.54 | 98.19 | 99.54 | 98.93 | 99.54 | 98.94 |
| | Lighter | 99.85 | 98.49 | 99.81 | 98.68 | 99.84 | 98.73 |
| *k* | Quake | 17 | 17 | 17 | 17 | 17 | 17 |
| | SOAPec | 17 | 17 | 17 | 17 | 17 | 17 |
| | Musket | 23 | 19 | 23 | 19 | 23 | 19 |
| | Bless | 31 | 23 | 31 | 23 | 31 | 23 |
| | Lighter | 23 | 19 | 23 | 19 | 23 | 19 |

*(Rows labeled k show the k-mer sizes selected for each tool and dataset)*

*Table 3-2 Occupancy (fraction of bits set) for Bloom filters A and B for various coverages*

| Coverage | α | Bloom A (%) | Bloom B (%) |
|---|---|---|---|
| 20× | 0.35 | 53.082 | 34.037 |
| 35× | 0.2 | 53.085 | 34.398 |
| 70× | 0.1 | 53.082 | 34.429 |
| 140× | 0.05 | 53.094 | 34.411 |
| 280× | 0.025 | 53.088 | 34.419 |

Here is the formal argument on why Lighter's accuracy is near-constant as the depth of sequencing K increases and its memory footprint is held constant. The basic idea is that as K increases, we adjust α in inverse proportion. That is, we hold αK constant. For concreteness, consider two scenarios: scenario I, where the total number of k-mers is $K_1$ and subsampling fraction is $\alpha_1$, and scenario II where the number is $K_2 = zK_1$ and subsampling fraction is $\alpha_2 = \alpha_1/z$.

### *Contents of Bloom filter A*

The occupancy of Bloom filter A, as well as the fraction of correct k-mers in A, are approximately the same in both scenarios. This follows from the fact that $\kappa'_c \sim \text{Pois}(\alpha K(1-\varepsilon)/G)$, $\kappa'_e \sim \text{Pois}(\alpha K\varepsilon/H)$, and αK, ε, G, and H are constant across scenarios. This is also supported by our experiments, as seen in the main body of the manuscript. Because the occupancy does not change, we can hold the Bloom filter's size constant while achieving the same false positive rate.

## Accuracy of trusted / untrusted classifications

Also, if a read position and its neighbors within k − 1 positions on either side are error-free, then the probability it will be called trusted does not change between scenarios. We mentioned that when α is small, $P(\alpha_1) \approx P(\alpha_1/z) = P(\alpha_2)$. We also showed that the false positive rate of the bloom filter is approximately constant between scenarios, so $P^*(\alpha_1) \approx P^*(\alpha_1/z) = P^*(\alpha_2)$. Thus, the thresholds $y_x$ will also remain unchanged. $p_c = (p(\kappa'_c \geqslant 1))/(p(\kappa_c \geqslant 1))$ is the probability a correct k-mer is in the subsample given that it was sequenced. $p_c = (1 - e^{-\frac{\alpha(1-\epsilon)K}{G}})/(1 - e^{-\frac{(1-\epsilon)K}{G}}) \approx 1 - e^{-\frac{\alpha(1-\epsilon)K}{G}}$, since (1 − ε)K/G is large. $p_c$ is constant across scenarios since αK, ε, and G are constant. Since $p_c$ is constant, the parameters of the $B_{e,x}$ distribution are constant and the probability a correct position will be called trusted is also constant.

Now we consider an incorrect read position. We ignore false positives from Bloom filter A for now. $p_e = p(\kappa'_e \geq 1)/p(\kappa_e \geq 1) = (1 - e^{-\frac{\alpha\epsilon K}{H}})/(1 - e^{-\frac{\epsilon K}{H}})$ is the probability an incorrect k-mer is in the subsample given that it was sequenced. Since εK/H is close to 0, $e^{-\epsilon K/H} \approx 1 - \epsilon$ K/H and $p_e \approx (\alpha\epsilon K/H)/(\epsilon K/H) = \alpha$. Say an incorrect read position is covered by x k-mers; if $B_{e,x}$ is a random variable for the number of k-mers overlapping the position that appear in Bloom filter A, then $B_{e,x} \sim \text{Binom}(x,p_e) \approx \text{Binom}(x,\alpha)$. The probability of falsely trusting a position is therefore: $p(B_{e,x} \geq y_x) = \sum_{i=y_x}^{x} \binom{x}{i} p_e^i (1-p_e)^{x-i} \approx \sum_{i=y_x}^{x} \binom{x}{i} \alpha^i (1-\alpha)^{x-i}$. If we omit the $(1 - \alpha)^{x-i}$ term in the sum, what remains is an upper bound, i.e. $\sum_{i=y_x}^{x} \binom{x}{i} \alpha^i (1-\alpha)^{x-i} \leq \sum_{i=y_x}^{x} \binom{x}{i} \alpha^i$. Since $\alpha_2 = \alpha_1/z$, the upper bound in scenario II is lower by a factor of at least 1/z relative to the upper bound in scenario I. So an upper bound on the probability of labeling an

incorrect position as trusted decreases by a factor of at least z. When K increases, the number of distinct test cases for incorrect positions increases by a factor of at most z. Thus, we expect the total number incorrect positions labeled as trusted to remain approximately constant.

When α is small, the false positive rate β may dominate the probability pe. In practice, however, the false positive rate is usually small enough that the probability of a incorrect position being labeled as trusted due to false positives is extremely low. For example, when k-mer length k = 17, the false positive rate of Bloom A $\approx$ 0.004, the threshold $y_{2k-1}$ = 6, and α = 0.05. In this situation, $p(B_{e,x} \geqslant y_x) \approx 5 \cdot 10^{-11}$.

The above is not an exhaustive analysis, since we have not examined the case where a read position is error-free but not all of its neighbors within k−1 positions on either side are error-free. In this case, whether the threshold is passed depends chiefly on the whereabouts of the nearby errors.

### Contents of Bloom filter B

Given the analysis in the previous section, we expect that the collection of k-mers drawn from the stretches of trusted positions in the reads will not change much across scenarios and, therefore, the contents of Bloom filter B will not change much. This conclusion is also supported by our experiments, as seen in the main body of the manuscript.

### 3.2.5 Quality score

A low base quality value at a certain position can force Lighter to treat that position as untrusted even if the overlapping k-mers indicate it is trusted. First, Lighter scans the first 1 million reads in the input, recording the quality value at the last position in each read. Lighter then chooses the fifth-percentile quality value; that is, the value such that 5% of the values are less than or equal to it, say $t_1$. Using the same idea, we get another fifth-percentile quality value, say $t_2$, for the first base for the first 1 million reads. When Lighter is deciding whether a position is trusted, if its quality score is less than or equal to $\min\{t_1, t_2 - 1\}$, then it is called untrusted regardless of how many of the overlapping k-mers appear in Bloom filter A.

### 3.2.6 Parallelization

As shown in Figure 3-1, Lighter works in three passes: (1) populating Bloom filter A with a k-mer subsample, (2) applying the per-position test and populating Bloom filter B with likely correct k-mers and (3) error correction. For pass 1, because $\alpha$ is usually small, most time is spent scanning the input reads. Consequently, we found little benefit in parallelizing pass 1. Pass 2 is parallelized by using concurrent threads to handle subsets of input reads. Because Bloom filter A is only being queried (not added to), we need not synchronize accesses to A. Accesses to B are synchronized so that additions of k-mers to B by different threads do not interfere. Since it is typical for the same correct k-mer to be added repeatedly to B, we can save synchronization effort by first checking whether the k-mer is already present and adding it (synchronously) only if necessary. Pass 3 is parallelized by using concurrent threads to handle subsets of the reads; since Bloom filter B is only being queried, we need not synchronize accesses.

## 3.3 Results

https://github.com/mourisl/Lighter_paper/blob/revision1/README.md describes the exact command lines used.

### 3.3.1 Simulated dataset

#### *Accuracy on simulated data*

We compared the performance of Lighter v1.0.2 with Quake v0.3 [60], Musket v1.1 [63], BLESS v0p17 [64] and SOAPec v2.0.1 [72]. We simulated a collection of reads from the reference genome for the K12 strain of Escherichia coli (NC_000913.2) using Mason v0.1.2 [73]. We simulated six distinct datasets with 101-bp single-end reads, varying average coverage (35×, 75× and 140×) and average error rate (1% and 3%). For a given error rate e we specify Mason parameters -qmb -qmb e/2-qme -qme 3e, so that the average error rate is e but errors are more common toward the 3′ end, as in real datasets.

We then ran all four tools on all six datasets, with results presented in Table 3-1. BLESS was run with the -notrim option to make the results more comparable. In these comparisons, a true positive (TP) is an instance where an error is successfully corrected, i.e. with the correct base substituted. A false positive (FP) is an instance where a spurious substitution is made at an error-free position. A false negative (FN) is an instance where we either fail to detect an error or an incorrect base is substituted. As done in previous studies [63], we report the following summaries:

$$\text{recall}=TP/(TP+NP),$$

$$\text{precision}=TP/(TP+FP),$$

$$\text{Fscore}=2\times\text{recall}\times\text{precision}/(\text{recall}+\text{precision}) \text{ and}$$

$$\text{gain} =(TP-FP)/(TP+FN).$$

Since these tools are sensitive to the choice of k-mer size, we tried several values for this parameter (17, 19, 23, 27 and 31) and picked the value yielding the greatest gain in the accuracy evaluation. The k-mer sizes chosen are shown in the bottom rows of Table 3-1. Note that SOAPec's maximum k-mer size is 27. We found that Quake crashed for k-mer sizes 23 and up.

Unlike the other tools, Quake both trims the untrusted tails of the reads and discards reads it cannot correct. BLESS also trims some reads (even in -notrim mode), but only a small fraction (0.1%) of them, which has only a slight effect on results. For these simulation experiments, we measure precision and recall with respect to all the nucleotides (even the trimmed ones) in all the reads (even those discarded). This tends to lead to higher precision but lower recall for Quake relative to the other tools.

Apart from Quake, SOAPec, Musket and Lighter achieve the highest precision. Lighter achieves the highest recall, F-score and gain in the experiments with 1% error, and is comparable to BLESS when the error rate is 3%.

For the Mason-simulated 1% error dataset, we found that Lighter's gain was maximized by setting the k-mer size to 23. We therefore fix the k-mer size to 23 for subsequent experiments, except where otherwise noted.

### *Caenorhabditis elegans simulation*

We performed a similar accuracy test as in the previous section, but using data simulated from the larger C. elegans genome, WBcel235 (Table 3-3). We used Mason to simulate a dataset of 101-bp single-end reads with a 1% error rate totaling 35× coverage. We again tried several values for the k-mer size parameter (19, 23, 27 and 31) and picked the value yielding the greatest gain in the accuracy evaluation. As for the E. coli experiment, Lighter had the greatest recall, F-score and gain.

*Table 3-3 Simulation results with C. elegans genome*

| | Quake | SOAPec | Musket | Bless | Lighter |
|---|---|---|---|---|---|
| Recall | 85.7 | 53.4 | 90.31 | 98.99 | 98.12 |
| Precision | 99.82 | 99.84 | 99.59 | 95.64 | 99.66 |
| F-score | 92.22 | 69.58 | 94.72 | 97.29 | 98.88 |
| Gain | 85.55 | 53.31 | 89.94 | 94.48 | 97.78 |
| k | 19 | 23 | 27 | 31 | 31 |

### *Scaling with depth of simulated sequencing*

We also used Mason to generate a series of datasets with 1% error, like those used in Table 3-1, but for 20×, 35×, 70×, 140× and 280× average coverage. We ran Lighter on each and measured final occupancies (fraction of bits set) for Bloom filters A and B. If our assumptions and scaling arguments are accurate, we expect the final occupancies of the Bloom filters to remain approximately constant for relatively high levels of coverage. As seen in Table 3-2, this is indeed the case.

### *Cardinality of Bloom filter B*

We also measured the number of correct *k*-mers added to table B. We used the Mason dataset with 70× coverage and 1% error rate. The *E. coli* genome has 4,564,614 distinct *k*-mers, and 4,564,569 (99.999%) of them are in table B.

### *Effect of ploidy on Bloom filter B*

We conducted an experiment like that in the previous section but with Mason configured to simulate reads from a diploid version of the E. coli genome. Specifically, we introduced heterozygous SNPs at 0.1% of the positions in the reference genome. Mason then sampled equal numbers of reads from both genomes, making a dataset with 70× average coverage in total. Of the 214,567 simulated k-mers that overlapped a position with a heterozygous SNP, table B held 214,545 (99.990%) of them at the end of the run. Thus, Lighter retained in table B almost the same fraction of the k-mers overlapping heterozygous positions (99.990%) as of the k-mers overall (99.999%).

Musket and BLESS both infer a threshold for the multiplicity of solid k-mers. In this experiment, Musket inferred a threshold of 10 and BLESS inferred a threshold of 9. All three tools use a k-mer size of 23. By counting the multiplicity of the k-mers overlapping heterozygous positions, we conclude that Musket would classify 214,458 (99.949%) as solid and BLESS would classify 214,557 (99.995%) as solid. So in the diploid case, it seems Lighter's ability to identify correct k-mers overlapping heterozygous SNPs is comparable to that of error correctors that are based on counting.

Diploidy is one example of a phenomenon that tends to drive the count distribution for some correct k-mers (those overlapping heterozygous variants) closer to the count distribution for incorrect k-mers. In the Discussion section we elaborate on other such phenomena, such as copy number, sequencing bias and non-uniform coverage.

### *Effect of varying α*

In a series of experiments, we measured how different settings for the subsampling fraction α affected Lighter's accuracy as well as the occupancies of Bloom filters A and B. We still use the datasets simulated by Mason with 35×, 70× and 140× coverage.

As shown in Figure 3-3 and Figure 3-4, only a fraction of the correct k-mers are added to A when α is very small, causing many correct read positions to fail the threshold test. Lighter attempts to 'correct' these error-free positions, decreasing accuracy. This also has the effect of reducing the number of consecutive stretches of k trusted positions in the reads, leading to a smaller fraction of correct k-mers added to B, and ultimately to lower accuracy. When α grows too large, the $y_x$ thresholds grow to be greater than k, causing all positions to fail the threshold test, as seen in the right-hand side of Figure 3-4. This also leads to a dramatic drop in accuracy as seen in Figure 3-3. Between the two extremes, we find a fairly broad range of values for α (from about 0.15 to 0.3) that yield high accuracy when the error rate is 1% or 3%. The range is wider when the error rate is lower.

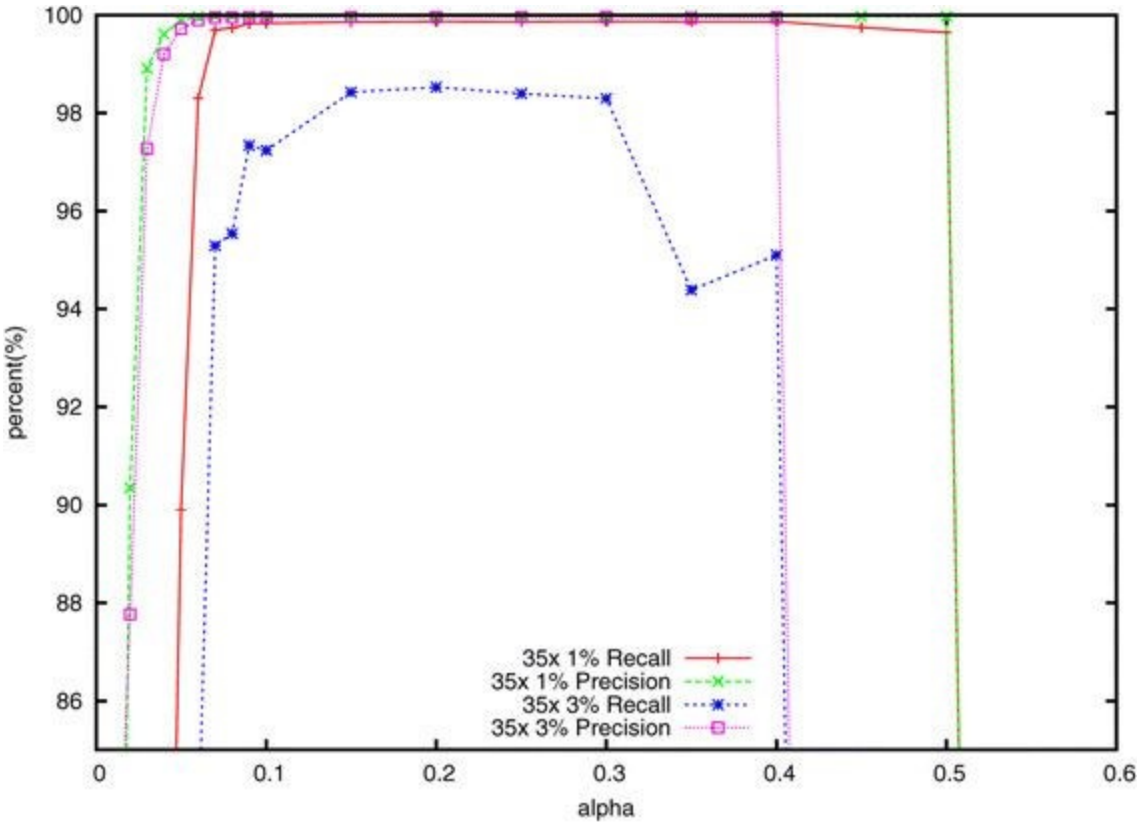



*Figure 3-3 The effect of α on the accuracy using the simulated 35× dataset*

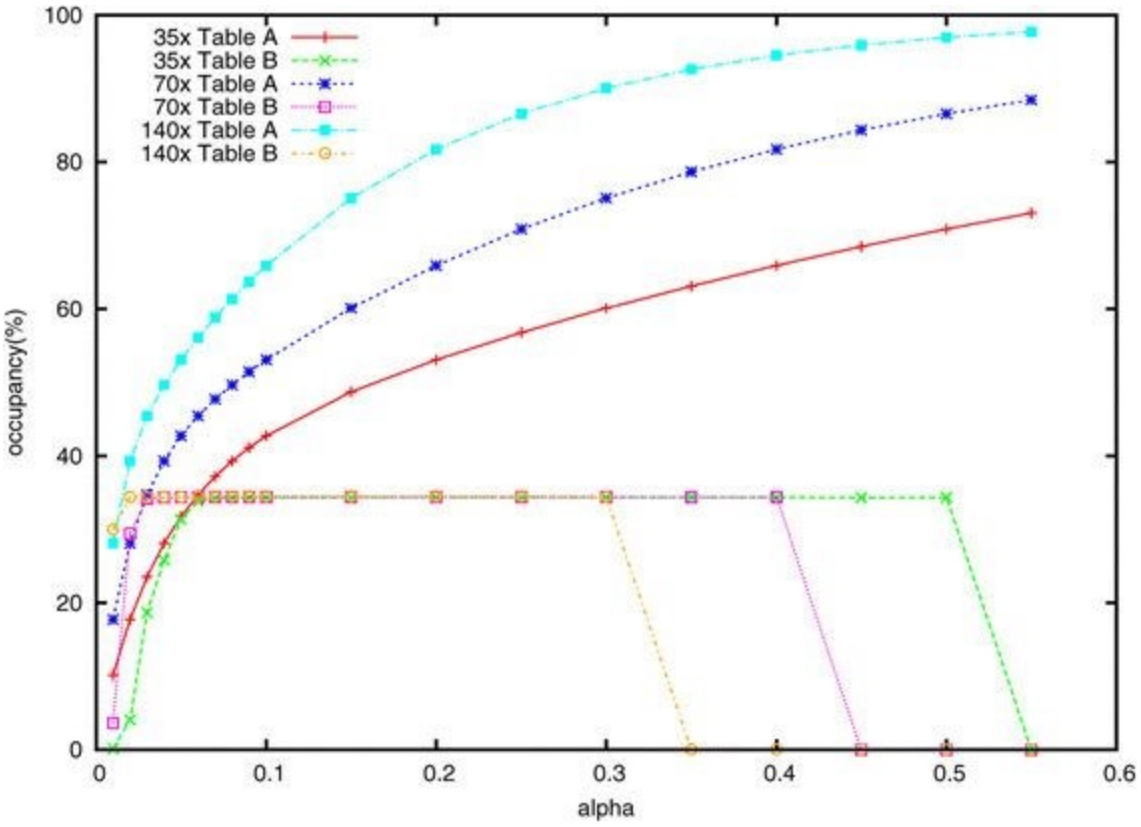



*Figure 3-4 The effect of α on occupancy of Bloom filters A and B. The effect of α on occupancy of Bloom filters A and B using simulated 35×, 70× and 140× datasets. The error rate is 1%.*

### *Effect of varying k*

A key parameter of Lighter is the k-mer length k. Smaller k yields a higher probability that a k-mer affected by a sequencing error also appears elsewhere in the genome. For larger k, the fraction of k-mers that are correct decreases, which could lead to fewer correct k-mers in Bloom filter A. We measured how different settings for k affect accuracy using the simulated data with 35× coverage and both 1% and 3% error rates. Results are shown in Figure 3-5. Accuracy is high for k-mer lengths ranging from about 18 to 30 when the error rate is 1%. But the recall drops gradually when the error rate is 3%.

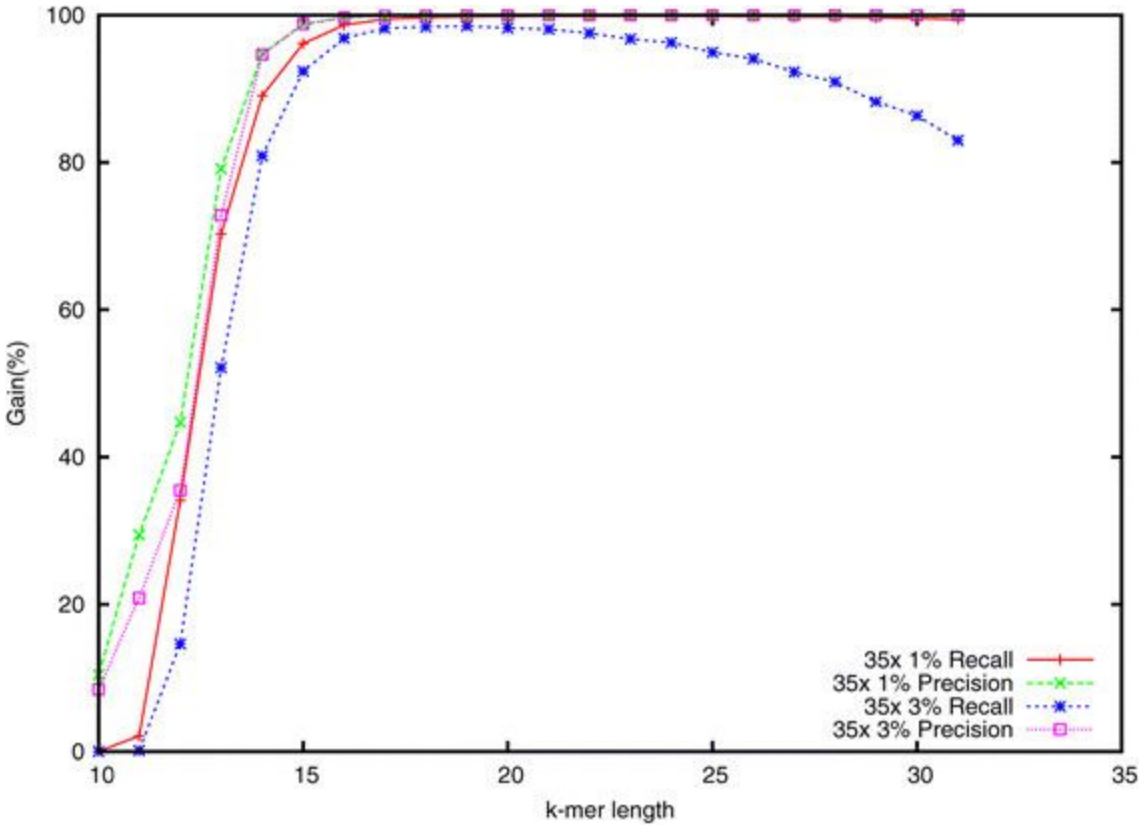



*Figure 3-5 The effect of k -mer length k on accuracy.*

## 3.3.2 Real datasets

### *Escherichia coli*

Next we benchmarked the same error correction tools using a real sequencing dataset, [EMBL-SRA ERR022075]. This is a deep DNA sequencing dataset of the the K-12 strain of the E. coli genome. To obtain a level of coverage more reflective of other projects, we randomly subsampled the reads in the dataset to obtain roughly 75× coverage (approximately 3.5 million reads) of the E. coli K-12 reference genome. The reads are 100 × 102 bp paired-end reads. Because BLESS cannot handle paired-end reads where the ends have different lengths, we truncated the last two bases from the 102-bp end before running our experiments. We again ran BLESS with the -notrim option.

These data are not simulated, so we cannot measure accuracy directly. But we can measure it indirectly, as other studies have [64], by measuring read alignment statistics before and after error correction. We use Bowtie2 [74] v2.2.2 with default parameters to align the original reads and the corrected reads to the E. coli K-12 reference genome. For each error corrector, we tested different k-mer sizes (17, 19, 23, 27 and 31) and chose the size that yielded the greatest total number of matching aligned nucleotides. For Quake and BLESS, we use only the reads (and partial reads) that remained after trimming and discarding for this evaluation. Results are shown in Table 3-4. Lighter yields the greatest improvement in fraction of reads aligned, whereas Quake and BLESS yield the greatest improvement in fraction of aligned bases that match the reference, with Lighter very close behind. As before, Quake is hard to compare to the other tools because it trims and discards many reads.

*Table 3-4 Alignment statistics for the 75× Escherichia coli dataset*

| | | Read level | | Base level | |
|---|---|---|---|---|---|
| | k | Mapped reads | Increase (%) | Matches/aligned base (%) | Increase (%) |
| Original | – | 3,464,137 | – | 99.038 | – |
| Quake | 19 | 3,373,498 | −2.62 | 99.659 | 0.63 |
| SOAPec | 17 | 3,465,819 | 0.05 | 99.13 | 0.09 |
| Musket | 17 | 3,467,875 | 0.11 | 99.601 | 0.57 |
| BLESS | 19 | 3,468,677 | 0.13 | 99.666 | 0.63 |
| Lighter | 19 | 3,478,658 | 0.42 | 99.639 | 0.61 |

*(k column shows k-mer size selected for each tool. First 'Increase' column shows percentage increase in reads aligned. Second 'Increase' column shows percentage increase in the fraction of aligned bases that match the reference genome. The original row is before error correction and the other rows are after error correction.)*

To assess accuracy further, we assembled the reads before and after error correction and measured relevant assembly statistics using Quast [75]. The corrected reads are those reported in Table 3-4. We used Velvet 1.2.10 [76] for assembly. Velvet is a De Bruijn graph-based

assembler designed for second-generation sequencing reads. A key parameter of Velvet is the De Bruijn graph's k-mer length. For each tool we tested different k-mer sizes for Velvet (43, 47, 49, 51, 53, 55, 57, 63 and 67) and chose the one that yielded the greatest NG50. We set the k-mer sizes of the error correctors to match those selected in the alignment experiment of Table 3-4. As before, we used only the reads (and partial reads) that remained after trimming and discarding for Quake and BLESS. For each assembly, we then evaluated the assembly's quality using Quast, which was configured to discard contigs shorter than 100 bp before calculating statistics. Results are shown in Table 3-5.

*Table 3-5 De novo assembly statistics for the Escherichia coli dataset*

| | N50 | NG50 | Edits/100 kbp | Misassemblies | Coverage (%) |
|---|---|---|---|---|---|
| Original | 94,879 | 94,879 | 3.41 | 0 | 97.496 |
| Quake | 89,470 | 88,209 | 11.62 | 4 | 97.515 |
| SOAPec | 98,111 | 94,879 | 3.49 | 1 | 97.473 |
| Musket | 86,421 | 86,421 | 6.45 | 0 | 97.53 |
| BLESS | 85,486 | 85,486 | 3.58 | 1 | 97.302 |
| Lighter | 105,460 | 105,460 | 3.71 | 1 | 97.477 |

N50 is the length such that the total length of the contigs no shorter than the N50 cover at least half the assembled genome. NG50 is similar, but with the requirement that contigs cover half the reference genome rather than half the assembled genome. Edits per 100 kbp is the number of mismatches or indels per 100 kbp when aligning the contigs to the reference genome. A misassembly is an instance where two adjacent stretches of bases in the assembly align either to two very distant or to two highly overlapping stretches of the reference genome. The Quast study defines these metrics in more detail [75].

Assemblies produced from reads corrected with the four programs are very similar according to these measures, with Quake and Lighter yielding the longest contigs and the greatest genome coverage. Surprisingly, the post-correction assemblies have more differences at nucleotide level compared to the pre-correction assemblies, perhaps due to spurious corrections.

### *GAGE human chromosome 14*

We also evaluated Lighter's effect on alignment and assembly using a dataset from the GAGE project [77]. The dataset consists of real 101 × 101 bp paired-end reads covering human chromosome 14 to 35× average coverage (approximately 36.5 million reads). For each error corrector, we tested different k-mer sizes (19, 23, 27 and 31) and chose the size that yielded the greatest total number of matching aligned nucleotides. For the assembly experiment, we set the k-mer size for each error corrector to match that selected in the alignment experiment. Also for each assembly experiment, we tested different k-mer sizes for Velvet (47, 53, 57, 63 and 67) and chose the one that yielded the greatest NG50.

The effect of error correction on Bowtie 2 alignment statistics are shown in Table 3-6. We used Bowtie 2 with default parameters to align the reads to an index of the human chromosome 14 sequence of the hg19 build of the human genome. As before, Lighter yields the greatest improvement in fraction of reads aligned, whereas Quake and BLESS yield the greatest improvement in fraction of aligned bases that match the reference, with Lighter very close behind.

*Table 3-6 Alignment statistics for the GAGE chromosome 14 dataset*

| | | Read level | Base level |
|---|---|---|---|

| | k | Mapped reads | Increase (%) | Matches/aligned base (%) | Increase (%) |
|---|---|---|---|---|---|
| Original | – | 35,993,147 | – | 98.507 | – |
| Quake | 19 | 32,547,091 | −9.57 | 99.845 | 1.36 |
| SOAPec | 19 | 36,116,405 | 0.34 | 98.768 | 0.26 |
| Musket | 19 | 36,316,699 | 0.9 | 99.109 | 0.61 |
| BLESS | 27 | 36,301,816 | 0.86 | 99.411 | 0.92 |
| Lighter | 19 | 36,320,688 | 0.91 | 99.235 | 0.74 |

We also tested the effect of error correction on *de novo* assembly of this dataset using Velvet for assembly and Quast to evaluate the quality of the assembly. For each tool we tested different k-mer sizes (19, 23, 27 and 31) and chose the one that yielded the greatest NG50. Results are shown in Table 3-7. Overall, Lighter's accuracy on real data is comparable to other error correction tools, with Lighter and BLESS achieving the greatest N50, NG50 and coverage.

*Table 3-7 De novo assembly statistics for the GAGE chromosome 14 dataset*

| | N50 | NG50 | Edits/100 kbp | Misassemblies | Coverage (%) |
|---|---|---|---|---|---|
| Original | 5,290 | 3,861 | 139.46 | 1263 | 78.778 |
| Quake | 4,829 | 3,520 | 141.59 | 1201 | 78.358 |
| SOAPec | 5,653 | 4,143 | 127.8 | 623 | 79.087 |
| Musket | 5,587 | 4,105 | 131.17 | 559 | 79.175 |
| BLESS | 5,898 | 4,345 | 128.4 | 581 | 79.279 |
| Lighter | 5,827 | 4,280 | 127.69 | 618 | 79.287 |

### *Caenorhabditis elegans*

Using the same procedure as in the previous section, we measured the effect of error correction on another large real dataset using the reads from accession [NCBI-SRA SRR065390]. Results are shown in Table 3-8 and Table 3-9. This run contains real 100 × 100 bp paired-end reads covering the C. elegans genome (WBcel235) to 66× average coverage (approximately 67.6

million reads). k-mer sizes for the error correctors and for Velvet were selected in the same way as for the chromosome 14 experiment. The alignment comparison shows BLESS achieving the greatest increase in fraction of reads aligned, and BLESS and Quake achieving the greatest fraction of aligned bases that match the reference, probably due to their trimming policy. Lighter does the best of the non-trimming tools in the alignment comparison. In the assembly comparison, Lighter and SOAPec achieve the greatest N50, NG50 and coverage.

*Table 3-8 Alignment statistics for the Caenorhabditis elegans dataset*

| | | Read level | | Base level | |
|---|---|---|---|---|---|
| | k | Mapped reads | Increase (%) | Matches/aligned base (%) | Increase (%) |
| Original | – | 63,017,855 | – | 99.048 | – |
| Quake | 19 | 60,469,150 | −4.04 | 99.834 | 0.79 |
| SOAPec | 19 | 63,032,768 | 0.02 | 99.185 | 0.14 |
| Musket | 23 | 63,060,601 | 0.07 | 99.42 | 0.38 |
| BLESS | 31 | 64,150,807 | 1.8 | 99.744 | 0.7 |
| Lighter | 23 | 63,081,655 | 0.1 | 99.469 | 0.43 |

*Table 3-9 De novo assembly statistics for the Caenorhabditis elegans dataset*

| | N50 | NG50 | Edits/100 kbp | Misassemblies | Coverage (%) |
|---|---|---|---|---|---|
| Original | 17,330 | 17,317 | 27.66 | 441 | 94.873 |
| Quake | 13,887 | 13,668 | 27.19 | 559 | 94.32 |
| SOAPec | 19,369 | 19,457 | 25.71 | 449 | 95.308 |
| Musket | 18,761 | 18,917 | 28.02 | 438 | 95.288 |
| BLESS | 17,673 | 17,693 | 29.24 | 524 | 94.968 |
| Lighter | 19,222 | 19,333 | 26.9 | 434 | 95.332 |

### 3.3.3 Speed, space usage, and scalability

We compared Lighter's peak memory usage, disk usage and running time with those of Quake, Musket and BLESS. These experiments were run on a computer running Red Hat Linux 4.1.2-52 with 48 2.1-GHz AMD Opteron processors and 512 GB memory. The input datasets are the same simulated E. coli datasets with 1% error rate discussed previously, plus the GAGE human chromosome 14 dataset and C. elegans dataset.

The space usage is shown in Table 3-10. BLESS and Lighter achieve constant memory footprint across sequencing depths. While Musket uses less memory than Quake, it uses more than either BLESS or Lighter. BLESS achieves constant memory footprint across sequencing depths, but consumes more disk space for datasets with deeper sequencing. Note that BLESS can be configured to trade off between peak memory footprint and the number of temporary files it creates. Lighter's algorithm uses no disk space. Lighter's only sizable data structures are the two Bloom filters, which reside in memory.

*Table 3-10 Memory usage (peak resident memory) and disk usage of error correction tools*

| | 35× | | 70× | | 140× | | chr14 | | Caenorhabditis elegans | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Mem | Disk | Mem | Disk | Mem | Disk | Mem | Disk | Mem | Disk |
| Quake | 2.8 GB | 3.3 GB | 7.1 GB | 6.0 GB | 14 GB | 12 GB | 48 GB | 57 GB | 86 GB | 99 GB |
| Musket | 119 MB | 0 | 165 MB | 0 | 225 MB | 0 | 1.4 GB | 0 | 2.5 GB | 0 |
| BLESS | 11 MB | 918 MB | 11 MB | 1.8 GB | 13 MB | 3.5 GB | 138 MB | 15 GB | 175 MB | 36 GB |
| Lighter | 35 MB | 0 | 35 MB | 0 | 35 MB | 0 | 514 MB | 0 | 514 MB | 0 |

*(Mem: memory)*

To assess scalability, we also compared running times for Quake, Musket and Lighter using different numbers of threads. For these experiments we used the simulated E. coli dataset with 70× coverage and 1% error. Results are shown in Figure 3-6. Note that Musket requires at least two threads due to its master–slave design. BLESS can only be run with one thread and its running time is 1,812 s, which is slower than Quake.

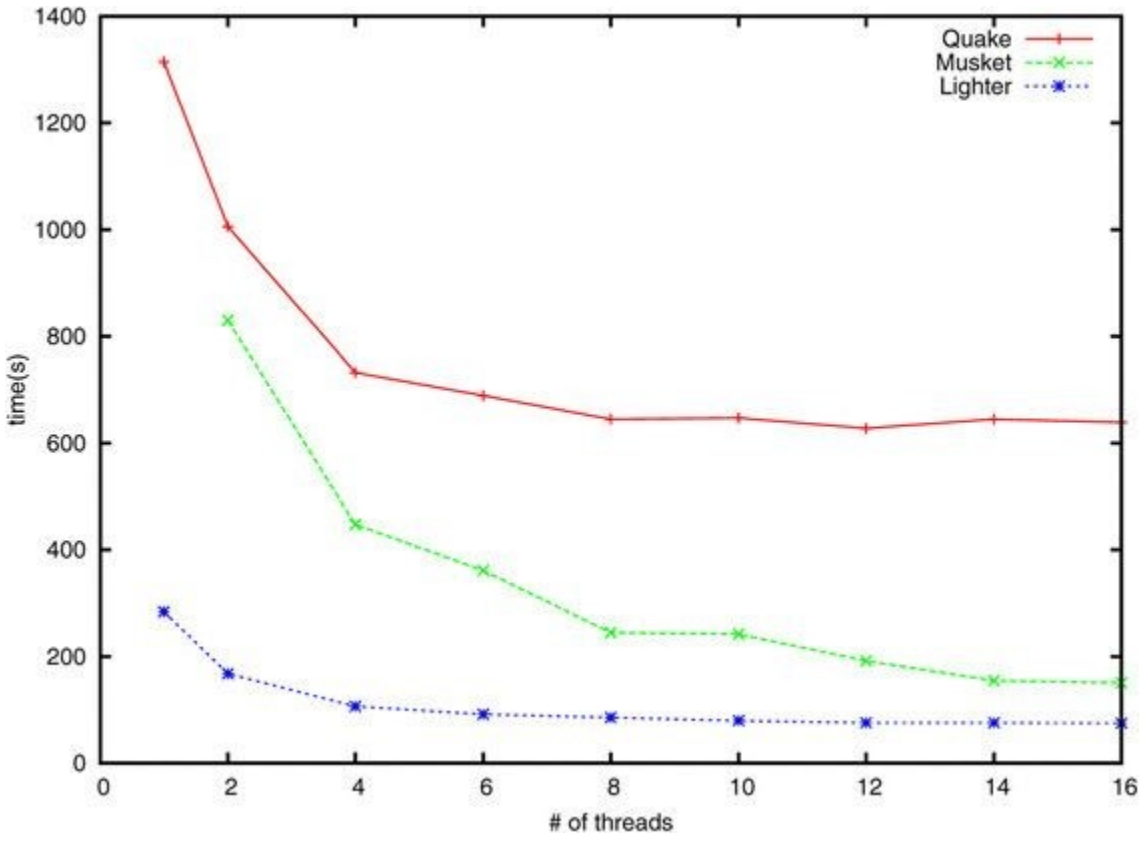



*Figure 3-6 Error correctors' running times. The running times for Quake, Musket and Lighter on 70× simulated dataset with increasing number of threads.*

## 3.4 Conclusions

At Lighter's core is a method for obtaining a set of correct k-mers from a large collection of sequencing reads. Unlike previous methods, Lighter does this without counting k-mers. By setting its parameters appropriately, its memory usage and accuracy can be held almost

constant with respect to depth of sequencing. It is also quite fast and memory-efficient, and requires no temporary disk space.

Though we demonstrate Lighter in the context of sequencing error correction, Lighter's counting-free approach could be applied in other situations where a collection of solid k-mers is desired. For example, one tool for scaling metagenome sequence assembly uses a Bloom filter populated with solid k-mers as a memory-efficient, probabilistic representation of a De Bruijn graph [67]. Other tools use counting Bloom filters [78, 79] or the related CountMin sketch [80] to represent De Bruijn graphs for compression [68] or digital normalization and related tasks [81]. We expect ideas from Lighter could be useful in reducing the memory footprint of these and other tools.

An important question is how Lighter's performance can be improved for datasets where coverage is significantly non-uniform, and where solid k-mers can therefore have widely varying abundance. In practice, datasets have non-uniform coverage because of ploidy, repeats and sequencing bias. Also, assays such as exome and RNA sequencing intentionally sample non-uniformly from the genome. Even in standard whole-genome DNA sequencing of a diploid individual, k-mers overlapping heterozygous variants will be about half as abundant as k-mers overlapping only homozygous variants. Lighter's ability to classify the heterozygous k-mers deteriorates as a result, as shown in the section Effect of ploidy on Bloom filter B above. Hammer [59] relaxes the uniformity-of-coverage assumption and favors corrections that increase the multiplicity of a k-mer, without using a threshold to separate solid from non-solid k-mers. A question for future work is whether something similar can be accomplished in

Lighter's non-counting regime, or whether some counting (e.g. with a counting Bloom filter [78, 79] or CountMin sketch [80]) is necessary.

A related issue is systematically biased sequencing errors, i.e. errors that correlate with the sequence context. One study demonstrates this bias in data from the Illumina GA II sequencer [82]. This bias boosts the multiplicity of some incorrect k-mers, causing problems for error correction tools. For Lighter, increased multiplicity of incorrect k-mers causes them to appear more often (and spuriously) in Bloom filters A and/or B, ultimately decreasing accuracy. It has also been shown that these errors tend to have low base quality and tend to occur only on one strand or the other [82]. Lighter's policy of using a fifth-percentile threshold to classify low-quality positions as untrusted will help in some cases. However, because Lighter canonicalizes k-mers (as do many other error correctors), it loses information about whether an error tends to occur on one strand or the other.

Lighter has three parameters the user must specify: the k-mer length k, the genome length G and the subsampling fraction α. While the performance of Lighter is not overly sensitive to these parameters (see Figure 3-3 and Figure 3-5), it is not desirable to leave these settings to the user. In the future, we plan to extend Lighter to estimate G, along with appropriate values for k and α, from the input reads. This could be accomplished with methods proposed in the KmerGenie [83] and KmerStream [71] studies.

Work on this project was supported in part by NSF grant ABI-1159078 to L.F. and IIS-1349906 and Sloan Research Fellowship to B.L.. Lighter is free open-source software released under the

GNU GPL license, and has been compiled and tested on Linux, Mac OS X and Windows computers. The software and its source are available from https://github.com/mourisl/Lighter.

# Chapter 4

# Rcorrector: efficient and accurate error correction for Illumina RNA-seq reads

## 4.1 Introduction

Next-generation sequencing of cellular RNA (RNA-seq) has become the foundation of virtually every transcriptomic analysis. The large number of reads generated from a single sample allow researchers to study the genes being expressed and estimate their expression levels, and to discover alternative splicing and other sequence variations. However, biases and errors introduced at various stages during the experiment, in particular sequencing errors, can have a significant impact on bioinformatics analyses.

Systematic error correction of whole-genome sequencing (WGS) reads was proven to increase the quality of alignment and assembly (Chapter 3), two critical steps in analyzing next-generation sequencing data. There are currently several error correction methods for WGS reads, classified into three categories [84]. K-spectrum based methods, which are the most popular of the three, classify a k-mer as trusted or untrusted depending on whether the number of occurrences in the input reads exceeds a given threshold. Then, for each read, low-frequency (untrusted) k-mers are converted into high-frequency (trusted) ones. Candidate k-mers are stored in a data structure such as a Hamming graph, which connects k-mers within a

fixed distance, or a Bloom filter. Methods in this category include Quake [60], Hammer [59], Musket [63], Bless [64], BFC [85], and Lighter [86]. Suffix tree and suffix array based methods build a data structure from the input reads, and replace a substring in a read if its number of occurrences falls below that expected given a probabilistic model. These methods, which include Shrec [54], Hybrid-Shrec [87] and HiTEC [55], can handle multiple k-mer sizes. Lastly, multiple sequence alignment (MSA) based methods such as Coral [56] and SEECER [88] cluster reads that share k-mers to create a local vicinity and a multiple alignment, and use the consensus sequence as a guide to correct the reads.

RNA-seq sequence data differ from WGS data in several critical ways. First, while read coverage in WGS data is largely uniform across the genome, genes and transcripts in an RNA-seq experiment have different expression levels. Consequently, even low-frequency k-mers may be correct, belonging to a homolog or a splice isoform. Second, alternative splicing events can create multiple correct k-mers at the event boundaries, a phenomenon that occurs only at repeat regions for WGS reads. In both of these cases, the reads would be erroneously converted by a WGS correction method. Hence, error correctors for WGS reads are generally not well suited for RNA-seq sequences [89].

There is so far only one other tool designed specifically for RNA-seq error correction, called SEECER [88], based on the MSA approach. Given a read, SEECER attempts to determine its context (overlapping reads from the same transcript), characterized by a hidden Markov model, and to use this to identify and correct errors. One significant drawback, however, is the large amount of memory needed to index the reads. Herein we propose a novel k-spectrum based method, Rcorrector (RNA-seq error CORRECTOR), for RNA-seq data. Rcorrector uses a flexible k-

mer count threshold, computing a different threshold for a k-mer within each read, to account for different transcript and gene expression levels. It also allows for multiple k-mer choices at any position in the read. Rcorrector only stores k-mers that appear more than once in the read set, which makes it scalable with large datasets. Accurate and efficient, Rcorrector is uniquely suited to datasets from species with large and complex genomes and transcriptomes, such as human, without requiring significant hardware resources. Rcorrector can also be applied to other types of data with non-uniform coverage such as single-cell sequencing, as we will show later. In the following sections we present the algorithm, first, followed by an evaluation of this and other methods on both simulated and real data. In particular, we illustrate and compare the impact of several error correctors for two popular bioinformatics applications, namely, alignment and assembly of reads.

## 4.2 Methods

### 4.2.1 De Bruijn graph

In a first preprocessing stage, Rcorrector builds a De Bruijn graph of all *k*-mers that appear more than once in the input reads, together with their counts. To do so, Rcorrector uses Jellyfish2 [61] to build a Bloom counter that detects *k*-mers occurring multiple times, and then stores these in a hash table. Intuitively, the graph encodes all transcripts (full or partial) that can be assembled from the input reads. At run time, for each read the algorithm finds the closest path in the graph, corresponding to its transcript of origin, which it then uses to correct the read.

### 4.2.2 Read error correction: the path search algorithm

As with any k-spectrum method, Rcorrector distinguishes among solid and non-solid k-mers as the basis for its correction algorithm. A solid k-mer is one that passes a given count threshold and therefore can be trusted to be correct. Rcorrector uses a flexible threshold for solid k-mers, which is calculated for each k-mer within each read sequence. At run time, Rcorrector scans the read sequence and, at each position, decides whether the next k-mer and each of its alternatives are solid and therefore represent valid continuations of the path. The path with the smallest number of differences from the read sequence, representing the likely transcript of origin, is then used to correct k-mers in the original read.

More formally, let u be a k-mer in read r and S(u,c) denote the successor k-mer for u when appending nucleotide c, with $c \in \{A,C,G,T\}$. For example, in Figure 4-1, S(AAGT,C)=AGTC, k=4. Let M(u) denote the multiplicity of k-mer u. To find a start node in the graph from which to search for a valid path, Rcorrector scans the read to identify a stretch of two or more consecutive solid k-mers, and marks these bases as solid. Starting from the longest stretch of solid bases, it proceeds in both directions, one base at a time as described below. By symmetry, we only illustrate the search in the 5'→3' direction.

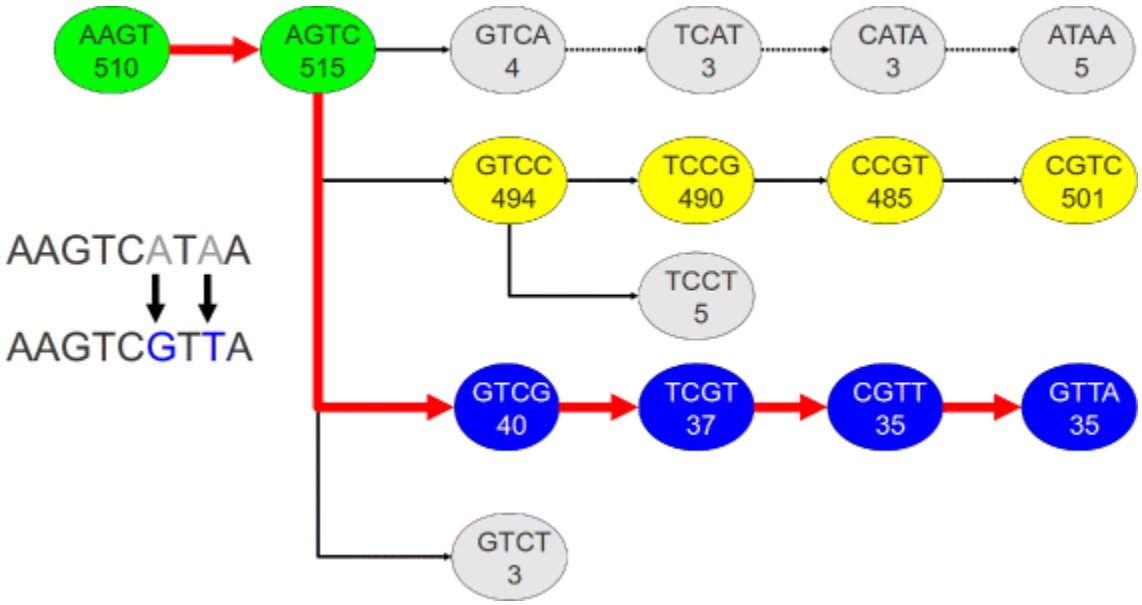



*Figure 4-1 Path extension in Rcorrector*

*(Four possible path continuations at the AGTC k-mer (k=4) in the De Bruijn graph for the r= AAGTCATAA read sequence. Numbers in the vertices represent k-mer counts. The first (top) path corresponds to the original read's representation in the De Bruijn graph. The extension is pruned after the first step, AGTC →GTCA, as the count M(GTCA)=4 falls below the local cutoff (determined based on the maximum k-mer count (494) of the four possible successors of AGTC). The second path (yellow) has higher k-mer counts but it introduces four corrections, changing the read into AAGTCCGTC. The third path (blue) introduces only two corrections, to change the sequence into AAGTCGTTA, and is therefore chosen to correct the read. The fourth (bottom) path is pruned as the k-mer count for GTCT does not pass the threshold. Paths 2 and 3 are likely to indicate paralogs and/or splice variants of this gene.)*

Suppose $u=r_i r_{i+1} \ldots r_{i+k-1}$ is the k-mer starting at position i in read r. Rcorrector considers all possible successors S(u,c), c∈{A,C,G,T}, and their multiplicities M(S(u,c)) and determines which ones are solid based on a locally defined threshold (see below). Rcorrector tests all the possible nucleotides for position i+k and retains those that lead to solid k-mers, and then follows the paths in the De Bruijn graph from these k-mers. Multiple k-mer choices are considered in order to allow for splice variants. If the nucleotide in the current path is different from $r_{i+k}$, then it is marked as a correction. When the number of corrections in the path exceeds an a priori defined threshold, Rcorrector terminates the current search path and starts a new one. In the end,

Rcorrector selects the path with the minimum number of changes and uses the path's sequence to correct the read. To improve speed, Rcorrector does not attempt to correct solid positions, and gradually decreases the allowable number of corrections if the number of searched paths becomes large.

### 4.2.3 A flexible local threshold for solid k-mers

Let u be the k-mer starting at position i in the read, as before. Unlike with WGS reads, even if the multiplicity M(S(u,$r_{i+k}$)) of its successor k-mer is very low, the base $r_{i+k}$ may still be correct, for instance sampled from a low-expression transcript. Therefore, an RNA-seq read error corrector cannot simply use a global k-mer count threshold. Rcorrector uses a locally defined threshold as follows. Let t= $\max_c$M(S(u,c)), calculated over all possible successors of k-mer u encoded in the De Bruijn graph. Rcorrector defines the local threshold at run time, f(t,r), as the smaller of two values, a k-mer-level threshold and a read-level one: f(t,r)= min(g(t),h(r)).

The k-mer-level threshold is defined as $g(t) = \alpha t + 6\sqrt{\alpha t}$, where α is a global variation coefficient. Specifically, α is determined for each dataset from a sample of 1 million high-count k-mers (multiplicities over 1,000), as follows. Given the four (or fewer) possible continuations of a k-mer, Rcorrector calculates a value equal to the ratio between the second highest and the highest multiplicities. Then, α is chosen as the smallest such value larger than 95 % of those in the sample. This criterion ensures that only k-mers that can be unambiguously distinguished from their alternates will be chosen; lowering this parameter value will reduce the stringency. Note that the k-mer-level threshold is the same for a k-mer in all read contexts, but differs by k-mer.

To calculate the read-level threshold, Rcorrector orders all k-mers in the read by decreasing multiplicities. Let x be the multiplicity before the first sharp drop (> 2-fold) in this curve. Rcorrector then uses h(r)=g(x) as the read-level threshold. Refinements to this step to accommodate additional lower-count paths are described below.

### 4.2.4 Refinements

#### *Clustered corrections*

Once a set of corrections has been determined for a read, Rcorrector scans the read and selectively refines those at nearby positions. The rationale for this step is that the likelihood of two or more clustered errors is very low under the assumed model of random sequencing errors, and the read may instead originate from a paralog. More specifically, let $u_i$ and $u_j$ be the k-mers ending at two positions i and j, with $j-i<k$, and $M(u_i)$ and $M(u_j)$ their multiplicities. To infer the source for the k-mer, Rcorrector uses the local read context and tests for the difference in the multiplicities of k-mers before correction. If the difference is significant, then it is a strong indication for a cluster of sequencing errors. Otherwise (i.e., if $0.5<M(u_i)/M(u_j)<2$), then the k-mers are likely to have originated from the same path in the graph, corresponding to a low-expression paralog, and the read is deemed to be correct. Rcorrector will revert corrections at positions i and j and then iteratively revisit all corrections within distance k from those previously reverted.

#### *Unfixable reads*

Rcorrector builds multiple possible paths for a read and in the end chooses the path with the minimum number of base changes. If the number of changes over the entire read or within any

window of size k exceeds an a priori determined threshold, the read is deemed ‘unfixable’. There are two likely explanations for unfixable reads: i) the read is correct, and originates from a low-expression transcript for which there is a higher-expression homolog present in the sample; and ii) the read contains too many errors to be rescued.

In the first case, Rcorrector never entered the true path in the graph during the extension, and hence the read was incorrectly converted to the high-expression homolog. To alleviate this problem, Rcorrector uses an iterative procedure to lower the read-level threshold h(r) and allow lower count k-mers in the path.

Specifically, Rcorrector looks for the next sharp drop in the k-mer multiplicity plot to define a new and reduced h(r), until there is no such drop or the number of corrections is within the set limits.

### *PolyA tail reads*

The presence of polyA tail sequences in the sample will lead to k-mers with mostly A or T bases. Because their multiplicities are derived from a mixture distribution from a large number of transcripts, these k-mers are ignored during the correction process. Rcorrector will consequently not attempt to correct such k-mers.

### *Paired-end reads*

With paired-end reads, Rcorrector leverages the k-mer count information across the two reads to improve the correction accuracy. In particular, it chooses the smaller of the two read-level thresholds as the common threshold for the two reads. In doing so, it models the scenario where the fragment comes from a low-expression isoform of the gene, with one of the reads

specific to this isoform and the other shared among multiple, higher-expression isoforms. In this case, the lower of the two read-level thresholds better represents the originating transcript.

## 4.3 Results

We evaluate Rcorrector for its ability to correct Illumina sequencing reads, both simulated and real. We include in the evaluation four other error correctors: SEECER (v0.1.3), which is the only other tool specifically designed for RNA-seq reads, as well as at least one representative method for each of the three classes of WGS error correction methods. These include Musket (v1.1) and BFC (r181) for k-spectrum, Hybrid-Shrec (Hshrec) for suffix tree and suffix array, and Coral (v1.4) for MSA-based methods. Since many tools are sensitive to the k-mer size k, we test different k-mer sizes for each tool where applicable and report the result that produces the best performance. We assess the impact of all programs on two representative bioinformatics applications, read alignment and read assembly. Lastly, we show that Rcorrector can be successfully applied to other types of data exhibiting non-uniform read coverage, such as single-cell sequencing reads.

### 4.3.1 Evaluation on simulated data

In a first test, we evaluated all programs on a simulated dataset containing 100 million 100 bp long paired-end reads. Reads were generated with FluxSimulator [19] starting from the human GENCODE v.17 gene annotations. Errors were subsequently introduced with Mason [73]; error rates were extracted from alignments of same-length Illumina Human Body Map reads. As in Chapter 3, we evaluate the accuracy of error corrections by inspecting how each base was

corrected. Let true positives (TP) be the number of error bases that are converted into the correct nucleotide; false positives (FP) the number of error-free bases that are falsely corrected; and false negatives (FN) the number of error bases that are not converted or where the converted base is still an error. We use the standard measures of Recall=TP/(TP+FN), Precision=TP/(TP+FP), and F-score=2∗Recall∗Precision/(Recall+Precision) to evaluate all methods. For each tool we test different k-mer sizes and report the result with the best F-score.

Accuracy values and performance measurements for the six error correctors are shown in Table 4-1. All programs were run on a 256 GB RAM machine with a 48-core 2.1 GHz AMD Opteron(TM) processor, with 8 threads. Here and throughout the manuscript, all measures are expressed in percentages. The overall sensitivity is below 90 % for all methods due to the large number of polyA reads generated by FluxSimulator, which are left unchanged. Rcorrector has the best overall performance by all measures, with 88 % sensitivity and greater than 99 % precision, followed closely by SEECER. Rcorrector is also virtually tied with BFC for the fastest method, and is among the most memory efficient. In particular, at 5 GB RAM for analyzing 100 million reads, it required 12 times less memory than SEECER and can easily fit in the memory of most desktop computers (Table 4-1).

*Table 4-1 Accuracy of the six error correction methods on the 100 million RNA-seq simulated reads*

| Program | k | Recall | Precision | F-score | Run time (min) | Memory (GB) |
|---|---|---|---|---|---|---|
| SEECER | 31 | 87.13 | 96.93 | 91.77 | 177 | 61 |
| HShrec | - | 69.53 | 31.74 | 43.58 | 13641 | 30 |
| Coral | 31 | 58.35 | 85.14 | 69.25 | 1391 | 81 |
| Musket | 27 | 78.24 | 96.9 | 86.58 | 152 | 4 |
| BFC | 27 | 80.45 | 97.91 | 88.32 | 111 | 6 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Rcorrector | 27 | 88.94 | 99.84 | 94.07 | 118 | 5 |

The difficulty of error correction is expected to vary with the expression level of transcripts. Correcting reads from low-expression transcripts is particularly challenging because the error-containing k-mers cannot be easily distinguished on the basis of frequency. To assess the performance of the various tools with transcript expression levels, we divide the simulated transcripts into low-, medium-, and high-expression groups based on their relative abundance A assigned by FluxSimulator (low, A<5e−7; medium, 5e−7<A<0.0001; and high, A>0.0001). The results of each tool on the three subclasses are shown in Table 4-2. Most tools perform well on the high-expression dataset, with the exception of Coral (low sensitivity) and Hshrec (low precision). However, the performance for all methods, especially sensitivity, drops for reads from low-expression transcripts. Rcorrector has the best or comparable sensitivity and precision for each of the three classes of transcripts. Both Rcorrector and SEECER are significantly more precise (>86 % in all categories) and more sensitive than methods designed for DNA reads, especially for reads from low-expression transcripts.

*Table 4-2 Accuracy of six error correction methods on 100 million simulated reads, by expression level of transcripts*

| Program | Recall | Precision | F-score |
|---|---|---|---|
| Low expression | | | |
| SEECER | 32.78 | 90.54 | 48.14 |
| HShrec | 24.77 | 0.81 | 1.56 |
| Coral | 31.88 | 64.6 | 42.69 |
| Musket | 13.88 | 33.94 | 19.71 |
| BFC | 25.18 | 58.37 | 35.19 |
| Rcorrector | 39.4 | 86.62 | 54.16 |
| Medium expression | | | |
| SEECER | 86.58 | 97.05 | 91.51 |

| | | | |
|---|---|---|---|
| HShrec | 70.57 | 19.57 | 30.64 |
| Coral | 89.07 | 85.12 | 87.05 |
| Musket | 72.02 | 92.16 | 80.86 |
| BFC | 89.12 | 96.88 | 92.84 |
| Rcorrector | 87.73 | 99.66 | 93.31 |
| High expression | | | |
| SEECER | 87.39 | 96.9 | 91.9 |
| HShrec | 69.22 | 41.67 | 52.02 |
| Coral | 47.59 | 85.17 | 61.06 |
| Musket | 80.5 | 98.53 | 88.61 |
| BFC | 77.47 | 98.35 | 86.67 |
| Rcorrector | 89.42 | 99.91 | 94.37 |

*(k-mer sizes used are those in Table 4-1)*

### 4.3.2 Real datasets

For a more realistic assessment, we applied the tools to three real datasets that vary in their sequencing depth, read length, amount of sequence variation, and application area (Table 4-3). These include a plant RNA-seq dataset (peach embryos and cotyledons; SRA accession SRR531865), a lung cancer cell line (SRA accession SRR1062943), and a lymphoblastoid cell line sequenced as part of the GEUVADIS population variation project (SRA accession ERR188021). We use these three sets to evaluate the performance of programs on real data, as well as to illustrate the effects of error correction on the alignment and assembly of RNA-seq reads. Summary statistics for all datasets are shown in Table 4-3 and the histograms inferring the variation coefficients are shown in Figure 4-2.

*Table 4-3 Summary of datasets included in the evaluation*

| Name | Reads | Read length (bp) | Aligned | Perfectly aligned |
|---|---|---|---|---|
| Simulated | 99,338,716 | 100 | 81,994,413 | 21,070,024 |
| Peach | 38,883,238 | 75 | 24,775,386 | 5,617,514 |
| Lung | 113,313,254 | 50 | 110,771,941 | 85,160,322 |
| Geuvadis | 65,015,656 | 75 | 59,130,806 | 26,468,128 |

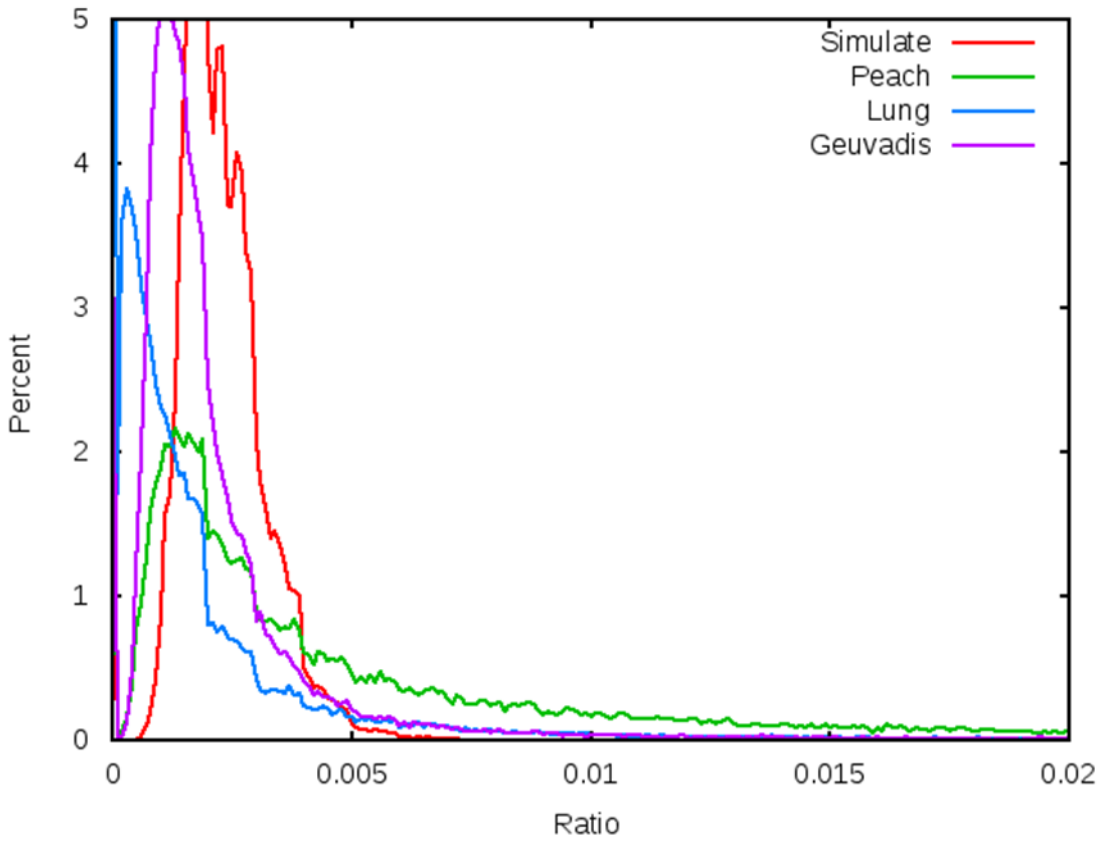



*Figure 4-2 Variation coefficient (α) for the 4 data sets*

*(For each data set (simulated, peach, Geuvadis and lung), we plot the histogram of variation ratios calculated as follows. The variation ratio for a k-mer is defined as the ratio between the second largest and the largest multiplicities among the four continuation k-mers. Hence, the lower the ratio, the more likely it is that the base change at the last position is a sequencing error. Conversely, values closer to 1 are indicative of polymorphisms, whereas middle values are potentially due to sequence differences between paralogs and/or isoforms of a gene. The histograms and distributions were estimated based on 1 million high-count k-mers. Then, we define α value corresponding to the 5$^{th}$ percentile of the distribution.)*

Unlike for simulated data, the ground truth for each base is unknown, making it impossible to judge performance directly and in an unbiased way. Instead, we use alignment rates to estimate the accuracy of error correction. We tested different k-mer sizes for each tool, and

chose the one maximizing the total number of matching bases. Statistics for alignments generated with Tophat2 (v2.0.13) [20] are summarized in Table 4-4. Lacking a true measure of sensitivity, the number and percentage of aligned reads as well as the per base match rate, as introduced in [3], are used to estimate sensitivity at read and base-level, respectively. The per base match rate is computed as the ratio of the total number of all the matching bases to the total number of aligned reads. Likewise, we introduce an alternate measure of specificity, defined as TN/(TN+FP), based on a high-confidence subset of the original reads (Table 4-4). We extracted those reads that have perfect alignments on the genome, i.e., that had exact sequence matches and the alignment of reads in a pair was concordant. These reads are expected to be predominantly error-free, therefore the proportion of reads that are not corrected represents a measure of specificity. As a caveat, these measures will falsely include those reads that are incorrectly converted to a paralog and aligned at the wrong location in the genome.

*Table 4-4 Tophat2 alignments of simulated and real reads*

| | k | Aligned | Observed rate | Base match rate | Specificity |
|---|---|---|---|---|---|
| Simulated reads | | | | | |
| Original | - | 81,994,413 | 82.54 | 99.391 | - |
| SEECER | 31 | 85,374,347 | 85.943 | 99.988 | 99.619 |
| Hshrec | - | 77,488,558 | 78.004 | 99.888 | 97.886 |
| Coral | 31 | 84,662,510 | 85.226 | 99.745 | 99.494 |
| Musket | 27 | 84,892,466 | 85.458 | 99.906 | 99.739 |
| BFC | 27 | 84,844,168 | 85.409 | 99.918 | 99.889 |
| Rcorrector | 27 | 85,033,277 | 85.599 | 99.986 | 99.97 |
| Peach | | | | | |
| Original | - | 24,775,386 | 63.717 | 99.198 | - |
| SEECER | 27 | 29,056,747 | 74.728 | 99.879 | 99.199 |
| Hshrec | - | 24,496,308 | 63 | 99.265 | 96.027 |

| | | | | | |
|---|---|---|---|---|---|
| Coral | 23 | 28,974,141 | 74.516 | 99.316 | 99.027 |
| Musket | 27 | 28,345,203 | 72.898 | 99.256 | 99.677 |
| BFC | 31 | 26,553,943 | 68.291 | 99.278 | 99.777 |
| Rcorrector | 23 | 30,563,388 | 78.603 | 99.833 | 99.628 |
| Lung | | | | | |
| Original | - | 110,771,941 | 97.757 | 99.717 | - |
| SEECER | 23 | 111,261,651 | 98.189 | 99.855 | 98.239 |
| Hshrec | - | 102,121,932 | 90.124 | 99.781 | 89.786 |
| Coral | 23 | 111,107,133 | 98.053 | 99.809 | 98.33 |
| Musket | 27 | 110,907,828 | 97.877 | 99.781 | 98.698 |
| BFC | 23 | 111,427,773 | 98.336 | 99.824 | 99.359 |
| Rcorrector | 23 | 111,198,587 | 98.134 | 99.83 | 99.599 |
| Geuvadis | | | | | |
| Original | - | 59,130,806 | 90.949 | 99.477 | - |
| SEECER | 23 | 61,514,024 | 94.614 | 99.837 | 98.53 |
| Hshrec | 23 | 51,669,686 | 79.473 | 99.709 | 87.924 |
| Coral | 23 | 61,399,007 | 94.437 | 99.717 | 98.049 |
| Musket | 23 | 60,450,316 | 92.978 | 99.652 | 97.9 |
| BFC | 23 | 61,870,897 | 95.163 | 99.775 | 98.79 |
| Rcorrector | 23 | 61,641,866 | 94.811 | 99.814 | 99.227 |

Error correction improves alignment rates by 1–11 %, depending on the dataset (Table 4-4). Note that alignment rates themselves differ with the amount of sequence variation and quality of the data. Rcorrector, SEECER, and BFC take turns in being the most sensitive across the four datasets. However, only Rcorrector and SEECER are consistently ranked among the top results in each category. Rcorrector has the highest or comparable specificity, greater than 99.2 %, in all cases.

We further assess the impact of error correction on improving *de novo* assembly of RNA-seq reads. We used the transcript assembler Oases [90] to assemble the reads a priori corrected with each of the methods. To evaluate the quality of the assembled transcripts, we aligned them to the reference genome with the spliced alignment program ESTmapper/sim4db [91],

retaining only the best match for each transcript. We use conventional methods and measures to evaluate the performance in reconstructing full-length transcripts [11]. Specifically, we define a match between a reference annotation transcript and the spliced alignment of an assembled transcript if and only if they have identical intron chains, whereas their endpoints may differ. We used the GENCODE v.17 annotations and the peach gene annotations (v1.1) obtained from the Genome Database for Rosaceae as the gold reference for the real datasets, respectively, and the subset of GENCODE transcripts sampled by FluxSimulator for the simulated data. The results, shown in Table 4-5, again indicate that SEECER, Rcorrector, and BFC have the most impact on improving the accuracy and quality of the assembled transcripts, and show comparable performance. Of note, these measures only capture full transcripts, whereas many of the transcripts in the sample will not have enough reads to be assembled fully.

*Table 4-5 Oases assembly of simulated and real reads*

| Program | Simulated | | Peach | | Lung | | Geuvadis | |
|---|---|---|---|---|---|---|---|---|
| | Recall | Precision | Recall | Precision | Recall | Precision | Recall | Precision |
| Original | 30.575 | 48.862 | 28.879 | 16.41 | 4.957 | 10.475 | 5.997 | 16.749 |
| SEECER | 36.698 | 52.181 | 29.752 | 16.116 | 4.944 | 10.174 | 6.162 | 16.639 |
| Hshrec | 23.334 | 47.417 | 26.132 | 13.85 | 3.608 | 11.459 | 4.266 | 19.101 |
| Coral | 35.039 | 51.942 | 29.784 | 15.881 | 4.934 | 10.174 | 6.17 | 16.372 |
| Musket | 33.845 | 47.769 | 28.76 | 15.991 | 4.92 | 10.577 | 5.846 | 16.901 |
| BFC | 34.789 | 50.579 | 29.633 | 16.211 | 5.018 | 10.498 | 6.166 | 16.509 |
| Rcorrector | 36.763 | 52.144 | 29.355 | 15.951 | 5.012 | 10.478 | 6.222 | 16.375 |

Figure 4-3 illustrates the spliced alignments of a 13 exon transcript at the MTMR11 (myotubularin related protein) gene locus (chr1:149,900,543-149,908,791) assembled with Oases from the simulated reads before and after correction. All methods missed the first intron, which was supported by six error-containing reads, but produced partial reconstructions of the

transcript, consisting of multiple contigs. While all error correctors improved upon the original reads, Rcorrector produced the most complete and compact assembly, with only three contigs, including one containing the full reconstruction of exons 1–12.


*Figure 4-3 Transcripts assembled from the original and error-corrected reads at the MTMR11 gene locus*

*(Rcorrector (bottom panel) improves upon the original reads and leads to the most complete reconstruction of the transcript)*

### 4.3.3 Single-cell sequencing

While Rcorrector was designed to correct RNA-seq reads, the method is also applicable to a wider range of problems where read coverage is non-uniform.

Single-cell sequencing has recently emerged as a powerful technique to survey the content and variation within an individual cell. However, PCR amplification of the input DNA introduces biases in read coverage across the genome. We compared Rcorrector with SEECER and the error correction module built into the assembly package SPAdes (3.1.0) [92]. The latter is based on the error corrector BayesHammer [93], which accounts for variable depth coverage. We applied all three methods to correct 29,124,078 E. coli K-12 MG1655 Illumina reads [92], then aligned the corrected reads to the E. coli K-12 genome with Bowtie2 [74] and assembled them

with SPAdes. We evaluated the alignment outcome as described earlier and separately used the package QUAST [75] to assess the quality of the resulting genome assemblies.

As seen in Table 4-6, Rcorrector results in the largest number of aligned reads, and is also the most specific among the methods. Surprisingly, the built-in SPAdes error corrector shows very low specificity (41.5 %), primarily arising from BayesHammer's trimming of end sequences for some reads. In contrast, SEECER has very high specificity but relatively low sensitivity, as the number of mapped reads was actually reduced after correction. Rcorrector shows both the highest sensitivity and the highest precision, and is therefore the best choice for this dataset.

*Table 4-6 Bowtie2 alignment of single-cell sequencing reads*

| | k | Aligned | Rate | Base match rate | Specificity |
|---|---|---|---|---|---|
| Original | - | 27,002,682 | 92.716 | 98.863 | - |
| SPAdes | - | 27,104,190 | 93.065 | 99.675 | 41.482 |
| SEECER | 27 | 26,937,652 | 92.493 | 99.507 | 99.553 |
| Rcorrector | 19 | 27,227,855 | 93.489 | 99.711 | 99.998 |

For assembly, both Rcorrector and SEECER lead to longer contigs and better genome coverage compared to the built-in corrector in SPAdes, while Rcorrector additionally produces the smallest number of misassemblies (Table 4-7). To conclude, Rcorrector can be effectively applied to correct single-cell DNA sequencing reads.

*Table 4-7 SPAdes assembly of single-cell sequencing reads*

| | NG50 | Misassembly | Edits/100 kbps | Genome coverage |
|---|---|---|---|---|
| Original | 105,623 | 1 | 6.57 | 95.054 |
| SPAdes | 109,876 | 2 | 7.52 | 94.903 |
| SEECER | 110,103 | 2 | 7.26 | 95.059 |

| | | | | |
|---|---|---|---|---|
| Rcorrector | 110,103 | 1 | 10.02 | 95.094 |

*(NG50 is the minimum contig length such that the total number of bases in contigs this size or longer represents more than half of the length of the reference genome)*

## 4.4 Conclusions

Rcorrector is the first k-spectrum based method designed specifically for correcting RNA-seq reads, and addresses several limitations in existing methods. It implements a flexible k-mer count threshold, to account for different gene and transcript expression levels, and simultaneously explores multiple correction paths for a read, to accommodate isoforms of a gene. In comparisons with similar tools, Rcorrector showed the highest or near-highest accuracy on all datasets, which varied in their amount of sequencing errors as well as polymorphisms. Also, with a small 5 GB memory footprint for a 100 million read dataset, it required an order of magnitude less memory than SEECER, the only other tool designed specifically for RNA-seq reads. Lastly, Rcorrector was the fastest of all methods tested, taking less than two hours to correct the simulated dataset. Therefore, Rcorrector is an excellent choice for large-scale and affordable transcriptomic studies in both model and non-model organisms.

Work on this project was supported in part by NSF grants ABI-1159078 and ABI-1356078 to L.F..

Rcorrector is available from https://github.com/mourisl/rcorrector.

# Chapter 5

# Rascaf: Improving Genome Assembly with RNA Sequencing Data

## 5.1 Introduction

Recent years have seen a tremendous increase in the number and diversity of sequenced genomes [94]. More than 13,000 eukaryotes have been sequenced or are in the process of sequencing, and more are planned including hundreds of plants and animals. Most model organisms have been sequenced under the umbrella of large genome projects undertaken by broad international consortia with the aim to create high-quality reference sequences [95, 96, 97, 98, 99, 100, 101]. In recent years, second-generation sequencing technologies have dramatically accelerated the pace of generating new genomes, as reduced sequencing costs along with increased access to sequencing have made it possible for groups and even individual investigators to sequence the genome of the species they study. Virtually all of these projects will produce draft versions of the genomes, in which the chromosomes are assembled into a relatively large number of contigs separated by gaps. Annotation software will then use the contigs, typically within groups of contigs with known order and gap sizes (scaffolds) or full chromosomes, as the substrate on which to identify genes.

During a typical genome assembly process, overlapping reads are first used to build contigs, then contigs are connected into larger scaffolds using order and orientation information from mate–pair reads. Mates are sequenced from the two ends of DNA fragments in a size-selected library, and their relative distance (insert size), order, and orientation on the originating DNA sequence can be estimated with relatively high accuracy. Repetitive regions in the genome pose a significant challenge to assembly algorithms. To be able to reconstruct these sequences, insert sizes need to exceed the length of the repeat to allow anchoring the assembly onto the nonrepetitive flanking regions. Therefore, a typical genome assembly project will require multiple insert-size libraries, spanning from 500 bp to 8 to 10 kb. There is a rich body of work in developing scaffolding algorithms based on mate pairs from whole-genome sequencing dating back to the assembly of the first sequenced eukaryotic genomes [102]. However, building the critical long-insert libraries is expensive and labor intensive.

Once a draft genome sequence is produced, the first and most crucial step in its analysis is finding the genes, which then provide the basis for downstream studies of gene function and variation. Deep RNA-seq has become the primary means to characterize the genes of a species, and there are already a number of high-performance tools for RNA-seq read analysis, including alignment and transcript assembly tools [103, 12, 104, 2, 14, 20, 1]. It also provides critical information about species-specific genes and alternatively spliced variants, including novel protein-coding genes and noncoding RNAs. Errors and gaps in the assembly can however interfere with correct gene and transcript annotation by fragmenting the genes, deleting or scrambling the exons, and by locally altering the gene's sequence [105]. Therefore, to aid

investigators in their gene studies, every effort must be made to improve the quality of the assembly, particularly in the gene regions.

Gene structures, in which introns may span thousands of bases, provide an effective way to increase the completeness and continuity of an assembly in silico. Several genome sequencing projects, starting with the human genome, have used gene information from independently generated expressed sequence tags and full-length messenger RNAs (mRNAs) to detect assembly errors or recruit additional contigs into the assembly [97, 106, 107, 108]. However, tools that could be systematically applied to any genome project and take advantage of the next-generation sequencing data being generated have been lacking. Traditional mate pair-based scaffolding methods [109, 110, 111, 112, 113] rely on a uniform read coverage of the genome and a statistically well characterized insert-size distribution and cannot be directly applied to RNA-seq reads. Only two tools have been recently developed that take advantage of next-generation RNA sequencing: L_RNA_scaffolder [114] applies the gene-based approach using *de novo* transcript assemblies generated with tools such as Trinity [115], whereas AGOUTI [116] employs the RNA-seq read alignments directly, in the context of known gene annotations, to detect new connections. However, *de novo* assembly of RNA-seq sequences as employed by L_RNA_scaffolder is challenging and error prone as well as time consuming. Chimeric transcript reconstructions can lead to incorrect scaffolds, and low-expression genes may be only partially reconstructed or missed entirely and therefore have limited impact.

We developed Rascaf (RnA-SCAFfolder), a novel tool that uses the alignments of RNA-seq reads to identify new contig connections in a fragmented genome and improves the completeness and accuracy of the genes and genome simultaneously. Rascaf uses an exon block graph to

simultaneously represent a gene and the underlying contig relationships and to determine a heaviest contig path. Suggested contig connections can then be optionally validated by database searches for cross-species complementary DNA (cDNA) and protein evidence. When evaluated on both simulated and real sequence data, and against similar tools, Rascaf was both more accurate and highly efficient and therefore can be effectively used to increase the quality of new genome assemblies of plants and animals. More specifically:

1. Rascaf simultaneously improves an assembly and its gene annotations, resulting in longer scaffolds, more accurate scaffolds, and more complete gene models.
2. It has higher or comparable accuracy to the best of the other tools for each application tested.
3. Rascaf is highly precise with only a handful of misassemblies introduced, has a small memory footprint, and runs in minutes on a regular workstation for a typical RNA-seq data set and genome.
4. Rascaf identified 1000 to 10,000 new contig connections in the draft genomes of several Fragaria species and of the Rosaceae pear (Pyrus communis L.), thus increasing their utility.
5. The program can be used with a single or with multiple RNA-seq data sets simultaneously.
6. An optional in silico validation step searches the predicted contig joins against external cDNA or protein databases for independent evidence.

## 5.2 Methods

Rascaf builds an improved assembly in two stages. Stage 1, implemented in the program rascaf, determines a set of possible contig connections based on continuity information from alignments of paired-end RNA-seq reads. Once a set of possible connections is determined, Stage 2, implemented in the program rascaf-join, uses the connections to scaffold the new assembly and to generate the new genome sequence. The general framework is illustrated in Figure 5-1, and the data structures and methods are described below.

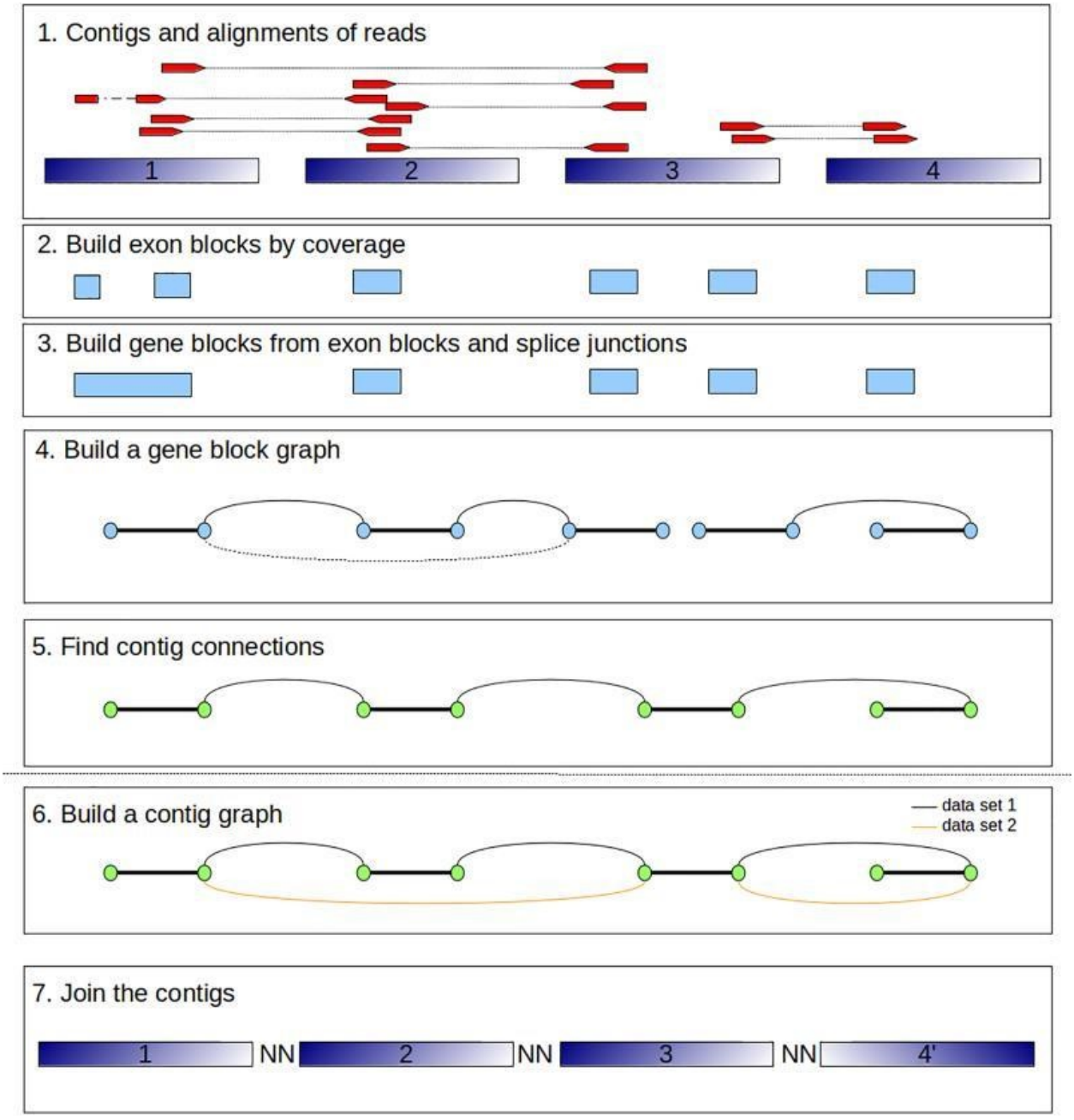



*Figure 5-1 Overall framework of the Rascaf algorithm*

*(Step 1: Prepare the raw assembly by splitting the scaffold-level assembly at runs of Ns. Paired-end RNA sequencing (RNA-seq) reads (red) connect four contigs (blue boxes) in the raw genome assembly. Step 2: Build the exon blocks by clustering read alignments along the genome. Step 3: Build the gene blocks by connecting exon blocks by introns extracted from spliced reads. Step 4: Build the gene block graph. Each gene block is represented by two nodes connected by a block edge (thick lines); ends of contig nodes linked by paired-end reads are then connected by mate*

*edges (thin lines). Continuous lines represent the selected block scaffolds along the heaviest path in the gene block graph, whereas dotted lines mark unselected edges in the graph. Step 5: Given a block scaffold determined above, find a set of candidate connections between contigs underlying the gene blocks. Steps 5 and 6: Build a contig graph by aggregating connections derived from multiple RNA-seq data sets. Each contig is represented by a pair of nodes connected by a contig edge (thick lines). Additionally, contigs adjacent in a scaffold in the raw assembly, or that were part of a contig connection detected in Step 5, are linked by a scaffold edge (thin lines). Step 7: Determine a set of cycle-free paths in the contig graph, using topological sorting, and use them to guide the construction of the new scaffolds.)*

### 5.2.1 Detecting contig connections

#### *Step 1*

The input to Rascaf is the draft genome in FASTA format and an alignment file of paired-end RNA-seq reads. If the assembled genome is in scaffolds, Rascaf first converts it to a contig-level (raw) assembly by splitting the sequences at runs of Ns.

#### *Steps 2 and 3*

The basic data structure employed by Rascaf is the exon block. An exon block denotes a maximal set of consecutive genomic coordinates covered by aligned RNA-seq reads corresponding to a block of overlapping exons. A gene block is an ordered set of exon blocks connected by spliced alignments corresponding to a portion of a gene located on the same contig.

#### *Step 4*

Rascaf builds a gene block graph as follows. Each gene block is represented by a pair of vertices (L, R) connected by an edge (block edge). When the two reads in a pair span different gene blocks, mate edges are added to connect the L endpoint of one gene block to the R endpoint of the other (Figure 5-2). This data structure is similar to the contig graph in [117]. One important

constraint on the gene block graph is that every path must alternate block and mate edges. Hence, setting the direction of one edge in a path will determine the directions of all remaining edges such that the concatenation of contig sequences along the path spells either the sequence of the genome or its reverse complement.

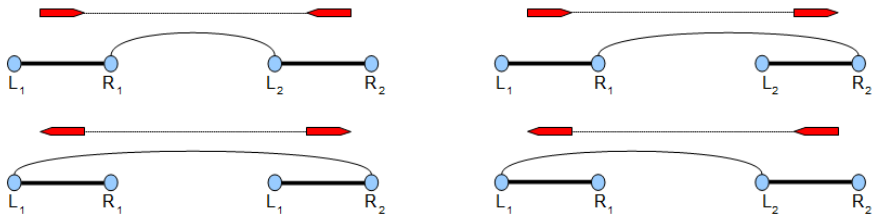



*Figure 5-2 Methods – finding contig connections (rascaf)*

*(There are four types of possible connections between two contigs ($L_1$,$R_1$) and ($L_2$,$R_2$) as dictated by the paired-end reads, represented by the mate edges below (thick lines). (1) Both contigs are in the forward orientation (1,2). (2) Contig 2 needs to be reversed (1,-2). (3) Contigs 1 and 2 must be swapped (2,1). (4) Contig 1 is reversed, and contigs 1 and 2 are swapped (2,-1). )*

Each component of the gene block graph corresponds to a gene or a portion of a gene. Rascaf employs a greedy method to find the order of the gene blocks in each component, choosing the most supported mate edge and then reiterating the search to extend the path in both directions. The procedure is terminated when there is no possible extension, on encountering a previously visited node, or at a sudden significant drop in read support. The algorithm produces a path of gene blocks, or block scaffold. If there are any remaining edges where neither of the adjacent nodes was selected, the procedure is repeated to find additional block scaffolds.

Sequencing and alignment errors may create false mate edges, leading to chimeric scaffolds. Rascaf uses alignment, read pair, and genomic context information to filter likely false positives. More specifically, Rascaf removes a mate edge if (i) it is supported by fewer than K reads (by

default, K = 2); (ii) there are multiple possible connections with similar support, indicating an ambiguous and potentially error prone connection; (iii) the concatenated sequence of exon blocks does not fit the mean and standard deviation of the insert size distribution for the RNA-seq reads; and (iv) the gene blocks appear to be duplicated in the assembly based on the overlap between their k-mer profiles, potentially indicating a paralogous connection.

#### *Step 5*

Once a set of block scaffolds is constructed, they are used as guides to find contig connections. Rascaf iteratively parses each block scaffold, starting from the one with the strongest read support, to create a list of contig connections and to decide the order and orientation of each contig within the scaffold. Connections that are incompatible with previously ordered contigs are ignored. In the end the procedure, implemented in the program rascaf, will determine a set of contig connections with known relative order and orientation.

### 5.2.2 Scaffolding guided by connections

#### *Step 6*

Once a set of contig connections is determined, rascaf-join incorporates them into a scaffolding algorithm to create a new assembled sequence. One ancillary benefit of separating the scaffolding from the detection of contig connections is that it allows combining multiple RNA-seq data sets, leveraging the variability in gene expression levels across the samples. For instance, the locus of a low-expression gene in one sample may be difficult to scaffold because connections here are hard to distinguish from noise, but this drawback can be mitigated when the gene is more richly covered in another data set.

Rascaf-join builds a contig graph that is similar in concept to the gene block graph described earlier. More specifically, each contig is represented by two vertices ($L_c$, $R_c$) connected by a contig edge. Two contigs are connected by a scaffold edge if they are adjacent in a scaffold in the raw assembly or are part of a contig connection identified by Rascaf. With multiple RNA-seq data sets, scaffold edges from different data sets could potentially introduce cycles. Rascaf-join detects any cycles in the contig graph using a depth-first search algorithm and removes all scaffold edges previously identified by rascaf that are adjacent to the contigs in the cycle to create an acyclic graph. It then attempts to improve each scaffold in the original assembly, starting from the longest, as described below.

Given a scaffold S in the raw original assembly, rascaf-join attempts to fill gaps in S and to extend it from both ends. Suppose S contains n contigs, with the associated contig nodes $L_1$, $R_1$,…,$L_n$ and $R_n$. Rascaf-join first finds the biconnected component in the contig graph containing all contigs from S (the biconnected component is a subgraph such that every node can be reached both from the path starting with $L_1 \rightarrow R_1$ and from the path starting with $R_n \rightarrow L_n$). Intuitively, a biconnected component contains the contigs from S as well as those contigs on a path that branches off and then returns to S. It then converts the component into a directed acyclic graph by fixing the path starting with $L_1 \rightarrow R_1$. Further, it uses a topological sort algorithm [118] to order the contigs in the connected component and to produce a longer scaffold S′ (Figure 5-3). In the end, Stage 2 generates a new assembly by recruiting additional contigs informed by the identified contig connections while adjusting the existing scaffolds as necessary to create more complete gene models.

While RNA-seq data present many advantages for genome scaffolding, it also has its drawbacks. For instance, the paired reads' inner distance along the genome, which may include introns of unknown sizes, cannot be characterized statistically, which can introduce ambiguity in the contig order and may lead to local rearrangements within a scaffold. This is especially problematic for genomes with very long introns and short contigs, in particular, with contigs located entirely within introns of the genes. Therefore, while using RNA-seq or gene structure information provides a highly practical solution to filling in the scaffold structure and building more complete gene models, further validation using, for instance, optical or physical maps and other data types may be necessary to resolve the local contig order at high resolution.

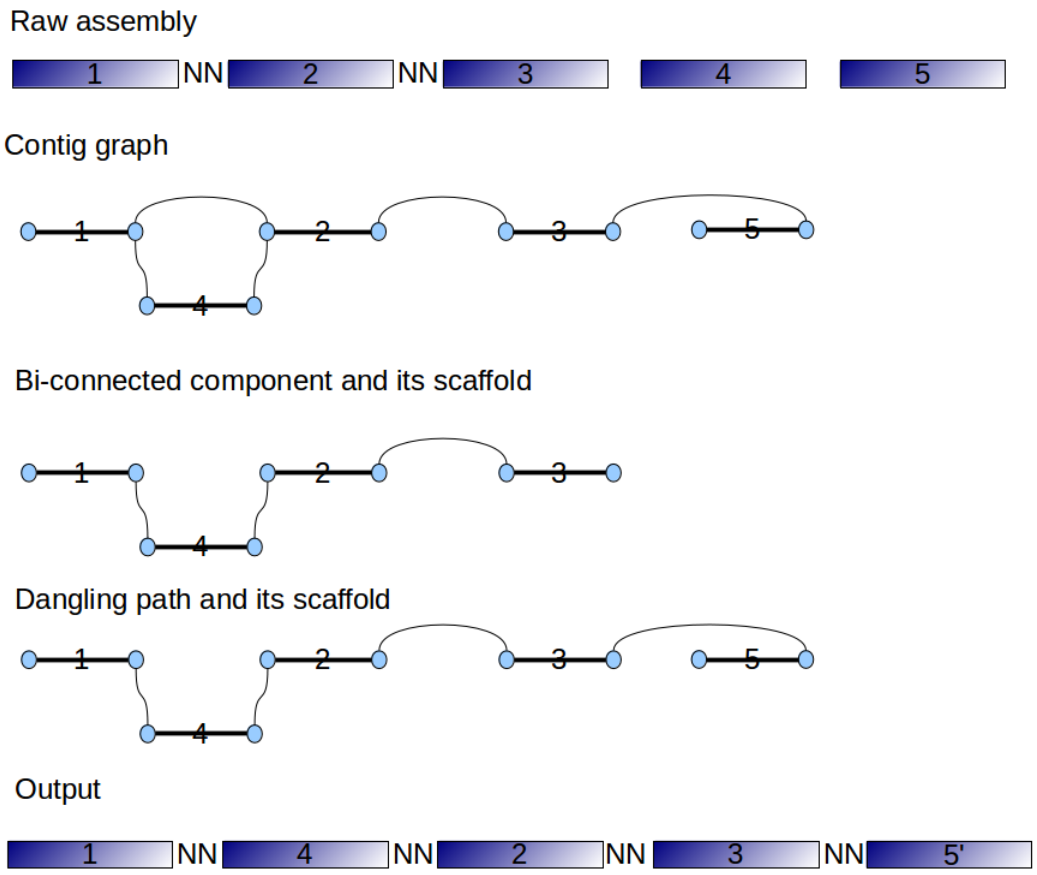



*Figure 5-3 Methods – scaffolding (rascaf-join)*

*(This example illustrates scaffolding starting from a raw assembly with 5 contigs (blue boxes): contigs 1-3 are connected into scaffold S, and contigs 4 and 5 are singletons. In stage 1, rascaf detects connections between contigs (1,4), (4,2) and (3,-5). The resulting contig graph has 5 (contig) nodes and 10 edges: 5 contig edges (thick lines), which connect the two nodes representing a contig, and 5 mate edges, representing either adjacency relationships in the original scaffold ((1,2) and (2,3)) or connections detected by rascaf based on RNA-seq paired-end reads ((1,4), (4,2) and (3,-5)). Starting from scaffold S, rascaf-join then traverses the contig graph to determine a bi-connected component and then uses a topological sort algorithm to determine a component path (scaffold) containing contigs 1,4,2 and 3 that satisfies all precedence relationships. Note that contig 5 is not part of the bi-connected component, as $L_5$ and R5 can be reached from $L_1 \rightarrow R_1$ but not from $R_3 \rightarrow L_3$ in scaffold S. We call $R_3, L_5, R_5$ a dangling path. To add dangling paths to the scaffold, rascaf-join traverses the component path starting from the last contig and moving backwards. In iteration i, it greedily chooses a maximal path starting with $Li \rightarrow Ri$ and trims it if it reaches a contig that has already been added to the scaffold. This path is then inserted in the component path (scaffold) following node $R_i$. Reverse dangling paths, which can be reached from $L_3 \rightarrow R_3$, are analyzed similarly.)*

## 5.3 Results

### 5.3.1 Performance evaluation on simulated control data

To obtain an accurate evaluation of performance on control data, we applied each program (Rascaf v.1.0.1; L_RNA_scaffolder (v. June 2013) and AGOUTI v.0.3.0-22) to a simulated data set. We generated an artificial genome by extracting the sequences of human chromosomes 1 and 12 and splitting them into contigs. For a more realistic model, we followed the contig size distribution of the Prunus persica v.1.0 genome from www.rosaceae.org [119]. In parallel, we used FluxSimulator [19] to generate 100 million 100-bp paired-end RNA-seq reads with average insert size 174 bp, respectively, using the GENCODE v.22 (www.gencode.org) gene annotations as reference. No chimeric reads were included. For Rascaf and AGOUTI, we mapped the ∼15 million reads from chromosomes 1 and 12 to the contigs with a fast-spliced aligner, HISAT [1]. Additionally, Rascaf has been adapted to incorporate partial alignments generated with BWA-mem [120], potentially obtained from spliced alignments spanning multiple contigs. For L_RNA_scaffolder, we first assembled the reads into transcripts with Trinity, and for AGOUTI we used as input annotation the gene models produced from the RNA-seq reads by Cufflinks [9].

To create a gold reference, we consider all pairs of ordered and oriented contigs in the assembly that are supported by read pairs. Let M be the size of this set and let N be the number of scaffold edges in the set of contig paths predicted by the program being evaluated. A contig pair ($c_1$,$c_2$) in the reference data set is said to be satisfied, or is a true positive (TP), if $c_1$ and $c_2$ appear in the same order and orientation in a contig path ($c_1$ and $c_2$ need not be adjacent). Conversely, a scaffold edge in a contig path is said to match the reference, or is a strong true

positive (STP), if its two contig nodes are connected by a read pair in the gold reference in the same order and orientation. With these concepts in place, we define sensitivity Sn = TP/M, and precision Pr = STP/N.

For this test case, Rascaf using both full and partial alignments had the best overall performance, at 0.763 sensitivity and precision 0.995, with L_RNA_scaffolder a very close second, at 0.741 sensitivity and precision 1.0 (Figure 5-4). This simple simulated example also illustrates one limitation when using RNA-seq reads directly without prior assembly, as implemented in Rascaf and AGOUTI, particularly for RNA-seq libraries with short insert sizes. Intuitively, a short insert size produces more read pairs sampled from the same exon and where one or both reads could span the boundary of the intron. Such cross-contig spliced alignments are missed by current alignment software. Indeed, both Rascaf's and AGOUTI's performance is improved with longer fragments (Table 5-1 Effects of library insert size on program performance, on the simulated dataTable 5-1), and their relative performance vis-à-vis the assembly-based L_RNA_scaffolder is fully recovered when incorporating partial (clipped) alignments produced with BWA-mem.

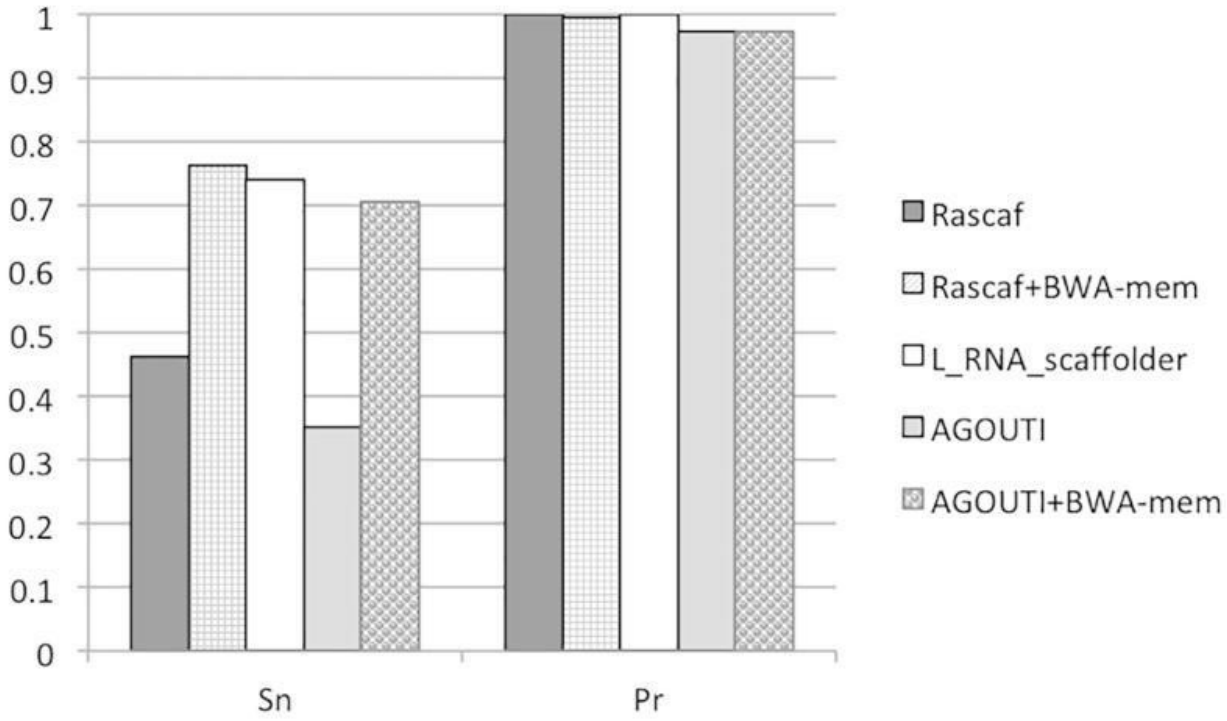



*Figure 5-4 Performance evaluation of programs on simulated data*

*Table 5-1 Effects of library insert size on program performance, on the simulated data*

| Program | m=174 bp | | m=225 bp | | m=275 bp | |
|---|---|---|---|---|---|---|
| | Sn | Pr | Sn | Pr | Sn | Pr |
| Rascaf | 0.463 | 1 | 0.596 | 1 | 0.656 | 1 |
| Rascaf+BWA-mem | 0.763 | 0.995 | 0.774 | 0.997 | 0.74 | 0.993 |
| L_RNA_scaffolder | 0.741 | 1 | 0.739 | 0.995 | 0.725 | 0.984 |
| AGOUTI | 0.352 | 0.974 | 0.498 | 0.989 | 0.573 | 0.987 |

### 5.3.2 In silico validation of Pyrus communis genome improvement

To assess the usefulness and accuracy of programs on a real assembly project, we applied them to improve the completeness and contiguity of the P. communis ('Bartlett') genome [121]. The pear genome has recently been sequenced using second-generation (Roche 454) single-end reads from 2- and 7-kb insert libraries, resulting in a 577 Mb assembly in 142,083 scaffolds

(184,520 contigs). Since no gold reference is available in this case, we performed an in silico validation by searching the concatenated gene sequences spanning each contig connection against the RefSeq gene database (BLASTn, dc-megablast option [122]. Note that since AGOUTI does not report the underlying gene structure, we could not include it in the evaluation. A match to a database homolog that spans the connection then greatly increases the confidence of the prediction. We queried each connection sequence and inspected the BLAST alignments for evidence of consistent coverage spanning the junction point. We deemed an alignment to be positive, and therefore provide proof for the connection, if all gene block sequences were contained in the alignment in the same order or orientation. We then distinguish between uncertain and potentially novel connections, in which alignments are compatible in order and orientation but cover only a subset of the gene blocks, and likely erroneous connections, which either show rearrangements between the gene block segments or portions of the segments (rearranged) or in which portions of the query sequence match different sequences in the database (chimeric). Note that both chimeric and negative connections may in fact reflect errors in other species' genomes or gene annotations rather than decision errors made by the tools. Lastly, since the programs may also report connections between the contigs from the same scaffold that are already known, we excluded any such connections from the performance measurements below.

When run with the SRR1609135 RNA-seq data set, comprised of 24.2 million 101-bp paired-end reads sampled from pear leaves, Rascaf produced 1286 and L_RNA_scaffolder generated 707 new putative connections (Table 5-2). Of these, 1218 (94.7%) of Rascaf connections were classified as positive, and an additional 55 (4.3%) were either uncertain or with no homolog in

the database and could potentially represent connections from novel genes or extensions of annotated genes. For L_RNA_scaffolder, 474 (67%) were validated, and an additional 152 (21.5%) were uncertain or unaligned. Only 13 (1.0%) Rascaf connections were chimeric or showed evidence of rearrangement compared with 81 (11.5%) for L_RNA_scaffolder. Therefore, Rascaf detected >2.5× as many positive (validated) connections than L_RNA_scaffolder and twice as many likely connections when the unaligned and uncertain cases were included, and reported four times fewer chimeric and rearranged (negative) cases. Hence, it was both more sensitive and more precise than L_RNA_scaffolder by a wide margin.

*Table 5-2 In silico validation of programs on the Pyrus communis genome by BLAST searches against the National Center for Biotechnology Information RefSeq mRNA database*

| Program | Validated | Uncertain | Unaligned | Rearranged | Chimeric |
|---|---|---|---|---|---|
| Rascaf | 1,218 | 42 | 13 | 4 | 9 |
| L_RNA_scaffolder | 474 | 128 | 24 | 38 | 43 |

An example of positively validated connection at the P. communis locus homologous to the Malus×domestica GDSL esterase/lipase At3g48460-like gene region (accession: XM_008395632.1) is shown in Figure 5-5A, and an uncertain connection at the P. ×bretschneideri peptidyl-prolyl cis-trans isomerase CYP71-like gene homolog (accession: XR_668155.1) missing support for its terminal 256 bp is shown in Figure 5-5B. For some of the connections classified as negative, the rearrangement resided in an existing contig rather than being introduced by our procedure (e.g., P. ×bretschneideri cryptochrome-1-like gene homolog, accession: XM_009380716.1; Figure 5-5C). Lastly, Figure 5-5D illustrates a chimeric connection as a result of a duplication within the M. ×domestica UDP-glycosyltransferase 74F1-like gene homolog (accession: XM_008396047). Therefore, Rascaf achieved >95% precision and can be

highly trusted to improve the sequence of a draft genome assembly while keeping the number of errors to a minimum.

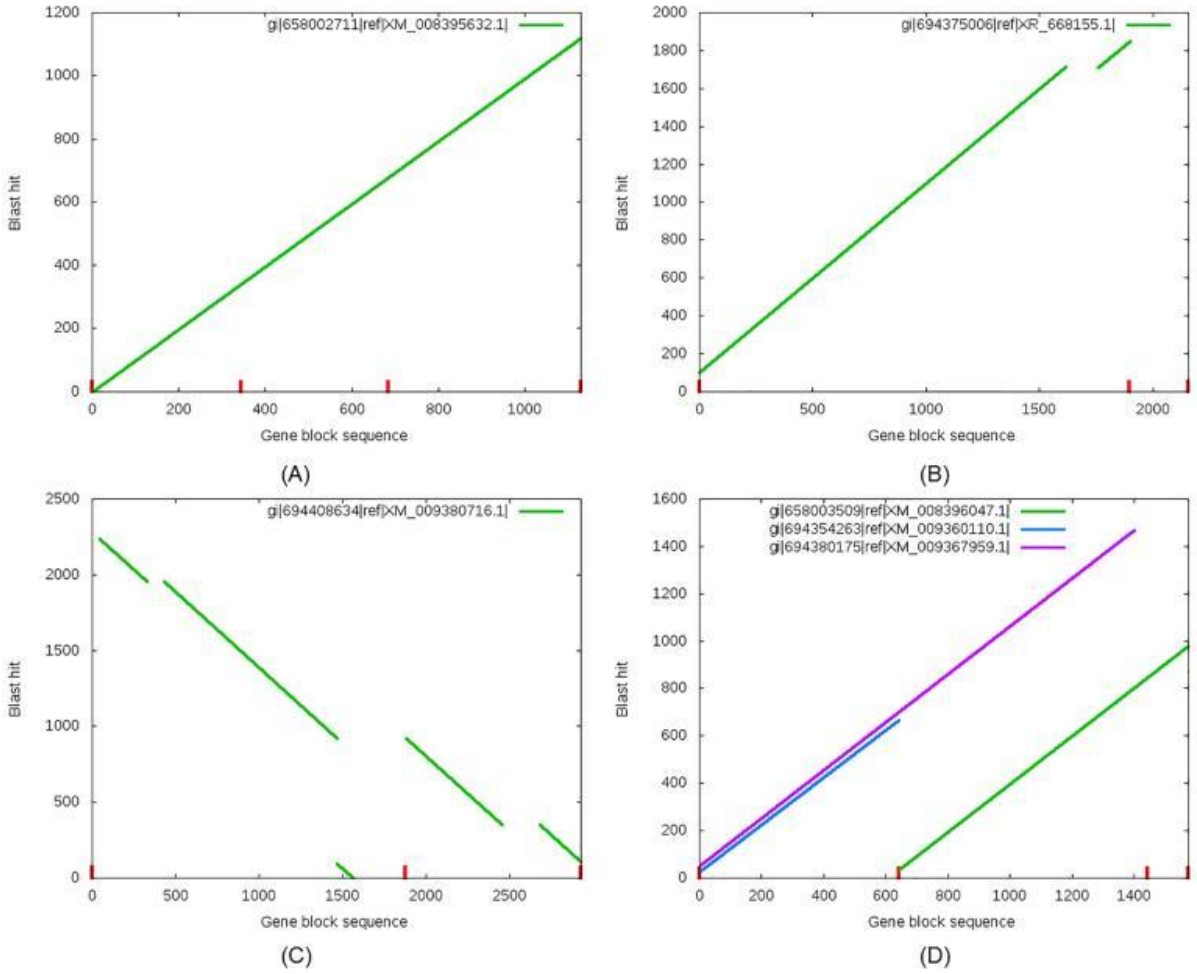



*Figure 5-5 Examples of in silico validation of contig connections detected in the Pyrus communis genome.*

*((A) Positive (validated) connection: alignments with the database homolog cover all gene blocks (marked by red tick marks along the horizontal axis) and are consistent in order and orientation. (B) Uncertain connection: alignments with the homolog do not cover the 256 bp in the second gene block. (C) Negative (incorrect) connection: alignments with the database homolog cover all gene blocks but are inconsistent in order and orientation. Here, however, the translocation is due to a misassembly within a contig of the original assembly. (D) Chimeric connection: alignments from three database homologs collectively cover all gene blocks. The chimeric construct here likely is due to the repetitive nature of the gene.)*

### 5.3.3 Assembly Improvement using Multiple RNA Sequencing Data Set

We further assessed the programs' performance on a high-quality genome system, which can serve as a more realistic control experiment. The 121-Mb genome of the model plant Arabidopsis thaliana Col-0 [95] has been reported and is one of the most extensively studied systems to date and therefore is the closest genome model yet to serving as a gold reference. As an additional goal, we sought to determine the feasibility and benefits of improving the assembly using multiple RNA-seq data sets simultaneously.

We used whole-genome DNA sequence data (SRA accession: SRR1810274; 60 million 100-bp reads) and assembled it with SOAPdenovo2 [72]. After filtering small scaffolds shorter than 500 bp, the draft genome assembly consisted of 37,948 contigs organized in 8082 scaffolds. We also downloaded 11 RNA-seq data sets sampled from plant leaves, root, and shoot apex (SRR2187604, SRR2080045, SRR971148, SRR1106559, SRR1187932, SRR1781769, SRR2060632, SRR2061405, SRR2895388, SRR2895627, and SRR2895761). For the multisample analysis, we added each data set to a growing pool to incrementally evaluate their impact on assembly improvement. Among the programs, only Rascaf can seamlessly integrate multiple RNA-seq data sets for analysis, potentially identifying and resolving internal conflicts. Nevertheless, we also ran L_RNA_scaffolder on the combined sets of transcripts assembled from the RNA-seq data. For AGOUTI, which was not designed to handle multiple data sets simultaneously, we ran the process iteratively (Table 5-3).

### 5.3.4 Comparative Evaluation of Programs

We first analyzed the performance of all programs by measuring the improvement on the assembly for a single RNA-seq data set (SRR2187604) using the Quast [75] evaluation software to compare against the reference A. thaliana genome. To calculate its statistics, Quast analyzes all scaffolds 500 bp or longer by aligning them to the reference genome with nucmer [123]. Rascaf found the most new connections, reducing the number of scaffolds by ∼400, with or without incorporating partial BWA-mem alignments. All three tools improved the NGA50, a measure of assembly continuity, by 2 to 4 kb (5–9%) as shown in Table 5-3 (top) (NGA50 is defined as the minimum length of a scaffold alignment such that 50% of the aligned portion of the assembly to the reference genome is in scaffold alignments this size or longer). At the same time, Rascaf and AGOUTI introduced a comparable number of misassemblies, while L_RNA_scaffolder was more imprecise. More in-depth analyses of the putative misassemblies introduced by the programs revealed that, in fact, most of these involved contigs that were misassembled in the original SOAPdenovo2 assembly. In several other cases, the reported misassembly was due to the intercontig gap length (1 kb) default parameter setting in Quast, which was too short to accommodate gaps potentially introduced by introns. After correcting for measurement errors and errors propagated from problematic contigs within the original assembly, the number of effective misassemblies introduced by each program was significantly reduced to 10 to 114 (0.8–8.8% of the total). Using Rascaf with additional partial BWA-mem matches did not bring significant improvement, likely because of the longer insert size in the RNA-seq library (245 bp) coupled with shorter exon sizes in A. thaliana. Overall, Rascaf

demonstrated better performance than the other tools, and both read alignment-based tools were considerably more precise than the transcript-based L_RNA_scaffolder.

*Table 5-3 Evaluation of Arabidopsis thaliana assemblies for single RNA sequencing (RNA-seq) data (top) and with 0, 1, …, 11 RNA-seq data sets (bottom), using Quast*

| Programs | Raw assembly | Rascaf | Rascaf+BWA-mem | L_RNA_scaffolder | AGOUTI |
|---|---|---|---|---|---|
| Single RNA-seq set (SRR2187604) | | | | | |
| Scaffolds | 8082 | 7686 | 7674 | 7759 | 7771 |
| NGA50 | 42,479 | 46,331 | 46,828 | 44,441 | 45,667 |
| Misassemblies | 1153 | 1180 | 1188 | 1296 | 1177 |
| Problematic scaffolds | 1412 | 1434 | 1434 | 1536 | 1433 |
| Effective misassemblies | na | 10 | 14 | 114 | 10 |
| Rascaf (multiple RNA-seq sets) | | | | | |
| Data sets | 0 (raw) | 1 | 2 | 6 | 11 |
| Scaffolds | 8082 | 7686 | 7626 | 7283 | 7222 |
| NGA50 | 42,479 | 46,331 | 46,898 | 49,673 | 50,571 |
| Misassemblies | 1153 | 1180 | 1190 | 1281 | 1302 |
| Problematic scaffolds | 1412 | 1434 | 1437 | 1473 | 1478 |
| Effective misassemblies | na | 10 | 14 | 66 | 77 |
| AGOUTI (iterative) | | | | | |
| Data sets | 0 (raw) | 1 | 2 | 6 | 11 |
| Scaffolds | 8,082 | 7,771 | 7,679 | 7,174 | 7,109 |
| NGA50 | 42,479 | 45,667 | 47,021 | 50,027 | 51,316 |
| Misassemblies | 1,153 | 1,177 | 1,209 | 1,401 | 1,417 |
| Problematic scaffolds | 1,412 | 1,433 | 1,439 | 1,450 | 1,454 |
| Effective misassemblies | - | 10 | 22 | 81 | 84 |
| L_RNA_scaffolder (batch) | | | | | |
| Scaffolds | 8,082 | 7,761 | 7,502 | 7,154 | 6,961 |
| NGA50 | 42,479 | 44,399 | 45,491 | 46,328 | 46,627 |
| Misassemblies | 1,153 | 1,296 | 1,459 | 1,731 | 1,896 |
| Problematic scaffolds | 1,412 | 1,536 | 1,618 | 1,772 | 1,870 |
| Effective misassemblies | - | 114 | 216 | 416 | 566 |

### 5.3.5 Performance Improvement with Multiple RNA Sequencing Data Set

In a second experiment, we assessed the impact of using multiple RNA-seq data sets on the completeness and accuracy of the resulting genome and its gene annotations. Since AGOUTI does not provide support for simultaneous analysis using multiple RNA-seq data sets, we ran the program iteratively, generating a new temporary assembly and improving it with the cumulative data set at the next step. As before, we used Quast to assess the impact on the assembled genome sequence when using 0 (raw assembly), 1, 2, 6, and 11 RNA-seq data sets. As shown in Table 5-3 (bottom) for Rascaf, the number of scaffolds and NGA50 values gradually improve as more data sets are added (by 5–10 and 9–19%, respectively). The number of effective misassemblies remains small (10–77), however, representing a growing fraction (0.8–5.9%) of the total assembly errors. AGOUTI was more aggressive in identifying new connections and produced slightly longer scaffolds at the expense of introducing more errors (Table 5-3), whereas L_RNA_scaffolder led to shorter scaffolds and a significantly larger (seven- to eight-fold) number of errors. We attribute Rascaf's high precision to its ability to identify and correct inconsistencies that arise from combining multiple RNA-seq data sets as implemented in rascaf-join.

Further, we assessed the impact of Rascaf's assembly improvement procedure on the accuracy and completeness of the gene repertoire. For each of the original and intermediate assemblies, we aligned the 35,215 A. thaliana RefSeq mRNA transcript sequences to the scaffolds using the spliced alignment programs ESTmapper and sim4db (Istrail et al., 2004; Walenz and Florea 2011). For each set, we plotted the percentage of transcript sequences that have more than f% of their bases in the primary alignment, for varying coverage levels f (f = 5, 10, …, 100). The

coverage curves are plotted in Figure 5-6. Adding RNA-seq data improves the coverage and the number of transcript sequences aligned at all coverage levels, especially at the higher coverage cutoffs. In particular, adding the first RNA-seq data set brings the number of transcripts with 90% coverage to 33,529 from 33,183 in the original assembly, and that number further increases to 33,847 when all 11 RNA-seq data sets are included. Most importantly, Rascaf increases the coverage for 506 to 991 genes across the 11 assemblies.

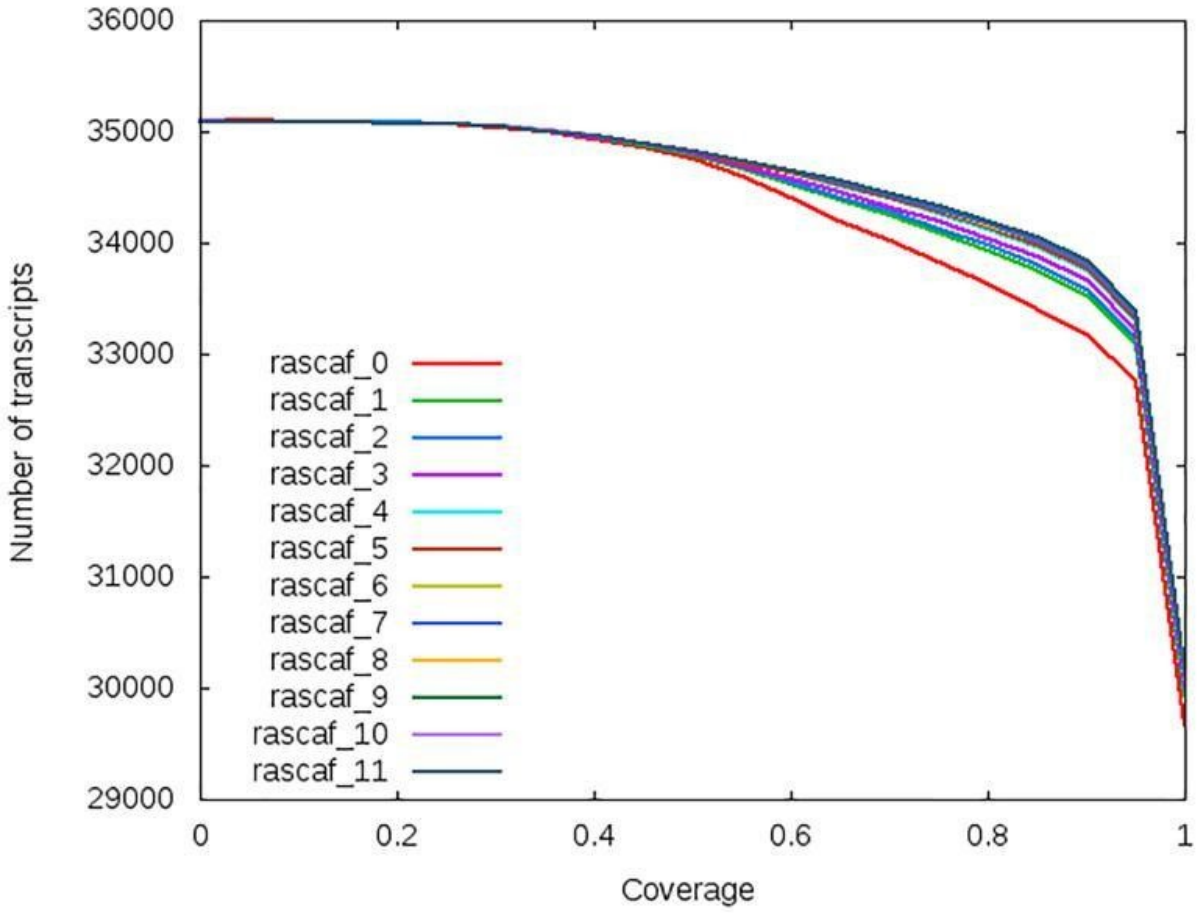



*Figure 5-6 Gene content evaluation of the improved Arabidopsis thaliana assemblies using 0, 1, … 11 RNA sequencing data sets*

*(Transcript coverage plots show the number of transcripts with a fraction x or more of their bases contained in the primary alignment of that transcript on the corresponding A. thaliana assembly, for coverage levels 0.05, 0.1, …, 1.0.)*

Therefore, using multiple RNA-seq sources improves both the assembled sequence and the gene annotations on a broader scale, albeit the benefits reach diminishing returns as more data sets recapitulating similar sets of genes are included.

### 5.3.6 Improving the Assembly and Gene Annotations of Sequenced Fragaria

Next-generation sequencing has dramatically accelerated the pace of genome sequencing projects, with dozens of new draft genomes being reported every few months. To illustrate the benefits of using RNA-seq to improve the quality of a genome assembly and its annotations, as implemented in Rascaf, we applied the method to several sequenced and in-progress Rosaceae. The octoploid genome of the cultivated strawberry (Fragaria × ananassa Duchesne ex Rozier) has been sequenced and draft assembled, along with those of four wild diploid relatives, F. iinumae Makino, F. nipponica Makino, F. nubicola (Hook. f.) Lindl. ex Lacaita, and F. orientalis Losinsk [124]. All genomes were sequenced with a combination of Illumina and Roche 454 technologies and assembled *de novo*. For our analyses, we downloaded up-to-date draft assemblies of these five Fragaria species from the Genome Database for Rosaceae (www.rosaceae.org) with quality indicators listed in Table 5-4.

*Table 5-4 Summary of quality indicators for the original (raw) assemblies of the sequenced Fragaria species*

| Assembly | No. of scaffolds | Total length (bp) | N50 |
|---|---|---|---|
| F. iinumae (v1.0) | 117,822 | 199,627,645 | 3309 |
| F. nipponica (v1.0) | 215,024 | 206,414,979 | 1275 |
| F. nubicola (v1.0) | 210,780 | 203,686,576 | 1291 |
| F. orientalis (v1.0) | 323,163 | 214,184,046 | 722 |
| F. × ananassa (v1.0) | 625,966 | 697,765,214 | 2201 |

We used available RNA-seq reads from F. × ananassa (SRA accession: ERR430941, 70 million 100-bp paired-end reads) and separately from the close relative F. vesca (SRA accession: SRR1930097, 76 million 101-bp paired-end reads). Fragaria vesca was previously sequenced and assembled with a combination of Illumina and 454 reads [107] and has undergone multiple rounds of curation to improve both the assembly and its gene models.

Rascaf found thousands of new putative connections for each genome, both when using the F. × ananassa and the cross-species F. vesca RNA-seq reads. We first assessed the likelihood and confidence of the connections using BLAST searches against the RefSeq mRNA database, as described above (Table 5-5). More than 96% of the connections found in each case could be validated, increasing to >99% likely connections when adding the uncertain and unaligned sequences, except for F. × ananassa improved with same-species RNA-seq reads, which had 89% positive validation rate and 98% likely connections rate.

*Table 5-5 In silico evaluation of predicted Rascaf connections in the Fragaria genomes by BLAST searches against the NCBI RefSeq mRNA database*

| Species | Validated | Uncertain | Unaligned | Rearranged | Chimeric |
|---|---|---|---|---|---|
| Rascaf with ERR430941 (F. × ananassa) | | | | | |
| F. iinumae | 5267 | 119 | 19 | 16 | 28 |
| F. nipponica | 7496 | 155 | 32 | 20 | 35 |
| F. nubicola | 8447 | 198 | 59 | 26 | 59 |
| F. orientalis | 10,147 | 279 | 112 | 16 | 72 |
| F. × ananassa | 5866 | 365 | 201 | 44 | 109 |
| Rascaf with SRR1930097 (F. vesca) | | | | | |
| F. iinumae | 1613 | 5 | 0 | 1 | 0 |
| F. nipponica | 2710 | 6 | 1 | 1 | 3 |
| F. nubicola | 3644 | 14 | 5 | 6 | 3 |
| F. orientalis | 3880 | 25 | 9 | 2 | 7 |
| F. × ananassa | 1876 | 36 | 9 | 6 | 8 |

We further assessed the impact on the completeness and accuracy of the gene annotations as an overall measure of assembly accuracy in gene regions. For this purpose, sequences of F. vesca RefSeq mRNA transcripts were mapped to each of the assemblies using ESTmapper and sim4db, and the portion of the bases contained in the primary alignment of each sequence was measured. The cumulative coverage plots were then computed as described in the previous section. Figure 5-7 shows the coverage plots for five of the species. For example, using ERR430941 leads to a significant increase (>5%) in the coverage of 6684 RefSeq transcripts in F. iinumae and 8967 in F. nipponica, with 3192 to 10,302 transcripts showing gains in coverage in the remaining species (Table 5-6). It also increases the numbers of complete or near-complete transcripts, defined as having 90% or more bases in a single alignment, from 10,800 to 14,562 (34.8% increase) for F. iinumae and from 5997 to 8918 (48.7% increase) for F. nipponica. These results are consistent with those in Table 5-5 and demonstrate that Rascaf significantly improves the completeness of genes in all five of these species.

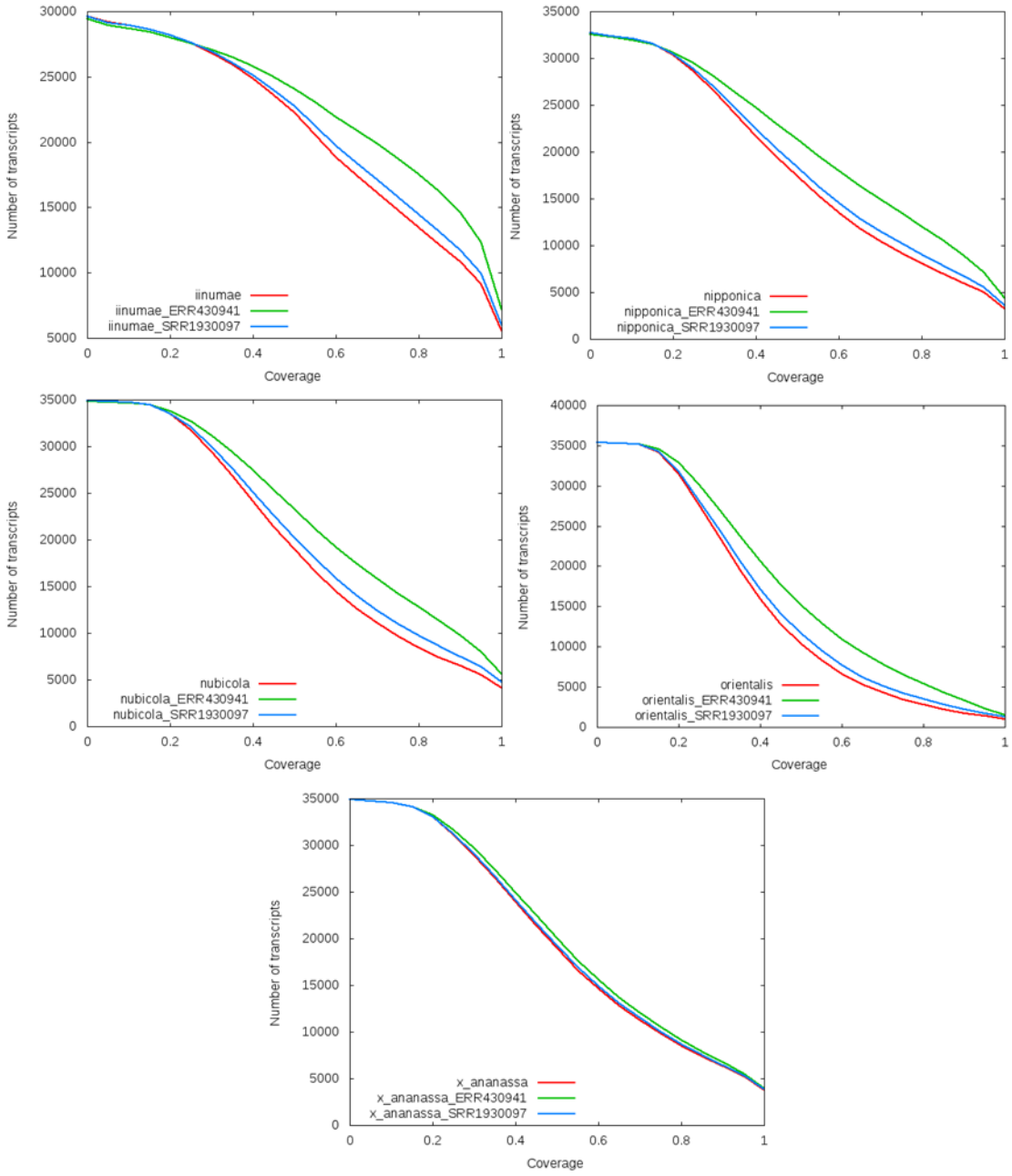



*Figure 5-7 Gene content evaluation of the Fragaria species assemblies before and after improvement with RNA-seq data*

*(Coverage plots show the number of transcripts with a fraction x or more contained in the primary alignment.)*

*Table 5-6 Evaluation of gene content for the five Fragaria assemblies before and after improvement using RNA sequencing (RNA-seq) data*

| Data set | Transcripts with >90% coverage | | Gain (>0) | Gain (>5%) |
|---|---|---|---|---|
| | Before | After | | |
| RNA-seq accession: ERR430941 | | | | |
| F. iinumae | 10,800 | 14,562 | 6931 | 6684 |
| F. nipponica | 5997 | 8918 | 9399 | 8967 |
| F. nubicola | 6468 | 9772 | 9928 | 9470 |
| F. orientalis | 1747 | 3262 | 11,033 | 10,302 |
| F. × ananassa | 6325 | 6711 | 3812 | 3192 |
| RNA-seq accession: SRR1930097 | | | | |
| F. iinumae | 10,800 | 11,699 | 1899 | 1816 |
| F. nipponica | 5997 | 6722 | 2782 | 2543 |
| F. nubicola | 6468 | 7495 | 3703 | 3441 |
| F. orientalis | 1747 | 2222 | 3687 | 3263 |
| F. × ananassa | 6325 | 6445 | 1019 | 834 |

*(Shown are the numbers of transcripts with 90% or more base coverage as well as the total number of transcripts that experienced gains in coverage.)*

### 5.3.7 Availability and Run-Time Considerations

The Rascaf package was developed using a combination of C++ and Perl modules. More specifically, source code for the two executables rascaf and rascaf-join was written in C++. A Perl module implementing the in silico BLAST evaluation against an external National Center for Biotechnology Information database, as an optional postprocessing step, is included in the package. The user can optionally invoke the search and modify it to point to a local or external database and to the BLAST search program of choice. Run times for analyses illustrated in this manuscript were roughly 15 min when using a single RNA-seq data set and <2 h for 11 RNA-seq samples when run sequentially on a machine with 512 GB RAM and AMD Opteron 2.1 GHz processor not including the HISAT alignment step. Memory used was <1 GB RAM in all cases, as alignment files are being read and processed sequentially. Running times for AGOUTI and

L_RNA_scaffolder alone were comparable; however, application of these tools in practice incurred additional run times for preassembling the RNA-seq reads into transcripts with Trinity (up to 1 d) and for generating the Cufflinks gene annotations. Therefore, Rascaf is highly memory and time efficient and can be readily used in assembly projects large and small and on a wide variety of platforms.

## 5.4 Conclusions

Fast and affordable next-generation sequencing has brought genome sequencing to the fingertips of groups and even individual researchers. However, assembling the short reads into a finished genome represents a significant challenge, demanding highly specialized expertise and sophisticated bioinformatics methods. Most of the genomes thus produced will be in draft form, consisting largely of ordered and oriented contigs grouped in scaffolds, but also many orphan contigs for which there may not be sufficient information to allow placement into scaffolds or chromosomes. We present a software tool, called Rascaf, that takes advantage of the continuity information from paired-end Illumina RNA-seq reads to identify new connections among contigs and use them to recruit additional contigs into an assembly and to improve the organization of scaffolds. As most genome sequencing projects also deep sequence the transcriptome, often of multiple tissues or organs, as part of gene annotation efforts, these RNA-seq sequences represent an abundant and readily available resource that can be effectively used to improve the genome sequence. Several genome sequencing projects have already used gene structure information to further help orient contigs and recruit additional contigs. However, these efforts largely used locally developed methods and relied on long cDNA

and mRNA sequences and more recently transcripts assembled *de novo* from Illumina reads, which are prone to introducing chimeric connections. These programs often rely on parameters, for instance to specify the criteria for alignments to be deemed significant, which need to be adjusted by the expert user. We present a practical and easy-to-use software tool, Rascaf, that can be universally used for any genome with no or very little need for calibration.

Rascaf improves the assembly and its gene annotations simultaneously, finding thousands of contig connections and contributing additional sequence to thousands of genes in the tested Rosaceae draft assemblies. It is also very precise, and almost every contig connection could be verified by independent evidence. Importantly, by separating the connection detection and scaffolding processes, Rascaf can incorporate multiple RNA-seq data sets, thus compensating for low-expression genes in any one sample while also eliminating likely errors that are revealed as incompatibilities. As an ancillary benefit, the split design reduces the end-to-end running time, allowing for parallelization of RNA-seq alignments across data sets as well as eliminating the overhead with building intermediate genome indices. Lastly, it provides a built-in mechanism for the user to intervene in the curation process; the user can manually inspect and edit the connections file to filter or add connections before processing them through the rascaf-join scaffolder. For convenience, scripts are provided to search and filter the connections against databases of proteins and cDNA sequences before scaffolding.

As another unique feature, Rascaf can incorporate partial alignment information, which is particularly beneficial for RNA-seq libraries with short insert sizes. Future work will assess incorporating alignments spliced across contigs, which cannot be generated by current alignment programs. Also, while Rascaf appears to work reasonably well with RNA-seq

sequences from a very close relative, the performance is likely limited, since current alignment tools are not equipped to handle sequence differences that arise from evolutionary changes, including block insertion–deletion mutations and evolutionary mutation patterns.

In comparisons with the only two other programs, Rascaf had higher or comparable accuracy in all tests and the lowest overall processing times. Fast and accurate, Rascaf is a highly practical and much needed addition to the current genome assembly and assembly curation compendium of tools.

Work on this project was supported in part by NSF grant IOS-1339134 to L.F. Rascaf is available free of charge for all distributed under a GNU General Public License and can be obtained from https://github.com/mourisl/Rascaf.

# Bibliography

[1] D. Kim, B. Langmead and S. L. Salzberg, "HISAT: a fast spliced aligner with low memory requirements," *Nature methods,* vol. 12, p. 357, 2015.

[2] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson and T. R. Gingeras, "STAR: ultrafast universal RNA-seq aligner," *Bioinformatics,* vol. 29, pp. 15-21, 2013.

[3] O. Kelemen, P. Convertini, Z. Zhang, Y. Wen, M. Shen, M. Falaleeva and S. Stamm, "Function of alternative splicing," *Gene,* vol. 514, pp. 1-30, 2013.

[4] J. Tazi, N. Bakkour and S. Stamm, "Alternative splicing and disease," *Biochimica et Biophysica Acta (BBA)-Molecular Basis of Disease,* vol. 1792, pp. 14-26, 2009.

[5] R. K. Singh and T. A. Cooper, "Pre-mRNA splicing in disease and therapeutics," *Trends in molecular medicine,* vol. 18, pp. 472-482, 2012.

[6] E. T. Wang, R. Sandberg, S. Luo, I. Khrebtukova, L. Zhang, C. Mayr, S. F. Kingsmore, G. P. Schroth and C. B. Burge, "Alternative isoform regulation in human tissue transcriptomes," *Nature,* vol. 456, p. 470, 2008.

[7] Q. Pan, O. Shai, L. J. Lee, B. J. Frey and B. J. Blencowe, "Deep surveying of alternative splicing complexity in the human transcriptome by high-throughput sequencing," *Nature genetics,* vol. 40, p. 1413, 2008.

[8] N. A. Fonseca, J. Rung, A. Brazma and J. C. Marioni, "Tools for mapping high-throughput sequencing data," *Bioinformatics,* vol. 28, pp. 3169-3177, 2012.

[9] 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, p. 511, 2010.

[10] M. Guttman, M. Garber, J. Z. Levin, J. Donaghey, J. Robinson, X. Adiconis, L. Fan, M. J. Koziol, A. Gnirke, C. Nusbaum and others, "Ab initio reconstruction of cell type--specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincRNAs," *Nature biotechnology,* vol. 28, p. 503, 2010.

[11] W. Li, J. Feng and T. Jiang, "IsoLasso: a LASSO regression approach to RNA-Seq based transcriptome assembly," in *International Conference on Research in Computational Molecular Biology*, 2011.

[12] A. M. Mezlini, E. J. M. Smith, M. Fiume, O. Buske, G. L. Savich, S. Shah, S. Aparicio, D. Y. Chiang, A. Goldenberg and M. Brudno, "iReckon: simultaneous isoform discovery and abundance estimation from RNA-seq data," *Genome research,* 2012.

[13] J. J. Li, C.-R. Jiang, J. B. Brown, H. Huang and P. J. Bickel, "Sparse linear modeling of next-generation mRNA sequencing (RNA-Seq) data for isoform discovery and abundance estimation," *Proceedings of the National Academy of Sciences,* vol. 108, pp. 19867-19872, 2011.

[14] L. D. Florea and S. L. Salzberg, "Genome-guided transcriptome assembly in the age of next-generation sequencing," *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB),* vol. 10, pp. 1234-1240, 2013.

[15] M. F. Rogers, J. Thomas, A. S. N. Reddy and A. Ben-Hur, "SpliceGrapher: detecting patterns of alternative splicing from RNA-Seq data in the context of gene models and EST data," *Genome biology,* vol. 13, p. R4, 2012.

[16] L. Florea, V. Di Francesco, J. Miller, R. Turner, A. Yao, M. Harris, B. Walenz, C. Mobarry, G. V. Merkulov, R. Charlab and others, "Gene and alternative splicing annotation with AIR," *Genome research,* vol. 15, pp. 54-66, 2005.

[17] L. Song and L. Florea, "CLASS: constrained transcript assembly of RNA-seq reads," in *BMC bioinformatics*, 2013.

[18] A. Ameur, A. Zaghlool, J. Halvardson, A. Wetterbom, U. Gyllensten, L. Cavelier and L. Feuk, "Total RNA sequencing reveals nascent transcription and widespread co-transcriptional splicing in the human brain," *Nature Structural and Molecular Biology,* vol. 18, p. 1435, 2011.

[19] T. Griebel, B. Zacher, P. Ribeca, E. Raineri, V. Lacroix, R. Guigó and M. Sammeth, "Modelling and simulating generic RNA-Seq experiments with the flux simulator," *Nucleic acids research,* vol. 40, pp. 10073-10083, 2012.

[20] 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, p. R36, 2013.

[21] L. Florea, L. Song and S. L. Salzberg, "Thousands of exon skipping events differentiate among splicing patterns in sixteen human tissues," *F1000Research,* vol. 2, 2013.

[22] J. Harrow, A. Frankish, J. M. Gonzalez, E. Tapanari, M. Diekhans, F. Kokocinski, B. L. Aken, D. Barrell, A. Zadissa, S. Searle and others, "GENCODE: the reference human genome annotation for The ENCODE Project," *Genome research,* vol. 22, pp. 1760-1774, 2012.

[23] A. Sveen, B. Johannessen, M. R. Teixeira, R. A. Lothe and R. I. Skotheim, "Transcriptome instability as a molecular pan-cancer characteristic of carcinomas," *BMC genomics,* vol. 15, p. 672, 2014.

[24] J. Eswaran, A. Horvath, S. Godbole, S. D. Reddy, P. Mudvari, K. Ohshiro, D. Cyanam, S. Nair, S. A. W. Fuqua, K. Polyak and others, "RNA sequencing of cancer reveals novel splicing alterations," *Scientific reports,* vol. 3, p. 1689, 2013.

[25] F. J. Miguel, R. D. Sharma, M. J. Pajares, L. M. Montuenga, A. Rubio and R. Pio, "Identification of alternative splicing events regulated by the oncogenic factor SRSF1 in lung cancer," *Cancer research,* 2013.

[26] H. Pimentel, M. Parra, S. L. Gee, N. Mohandas, L. Pachter and J. G. Conboy, "A dynamic intron retention program enriched in RNA processing genes regulates gene expression during terminal erythropoiesis," *Nucleic acids research,* vol. 44, pp. 838-851, 2015.

[27] H. Jung, D. Lee, J. Lee, D. Park, Y. J. Kim, W.-Y. Park, D. Hong, P. J. Park and E. Lee, "Intron retention is a widespread mechanism of tumor-suppressor inactivation," *Nature genetics,* vol. 47, p. 1242, 2015.

[28] H. Tilgner, D. G. Knowles, R. Johnson, C. A. Davis, S. Chakrabortty, S. Djebali, J. Curado, M. Snyder, T. R. Gingeras and R. Guigó, "Deep sequencing of subcellular RNA fractions shows splicing to be predominantly co-transcriptional in the human genome but inefficient for lncRNAs," *Genome research,* vol. 22, pp. 1616-1625, 2012.

[29] S. Djebali, C. A. Davis, A. Merkel, A. Dobin, T. Lassmann, A. Mortazavi, A. Tanzer, J. Lagarde, W. Lin, F. Schlesinger and others, "Landscape of transcription in human cells," *Nature,* vol. 489, p. 101, 2012.

[30] T. Steijger, J. F. Abril, P. G. Engström, F. Kokocinski, M. Akerman, T. Alioto, G. Ambrosini, S. E. Antonarakis, J. Behr, P. Bertone and others, "Assessment of transcript reconstruction methods for RNA-seq," *Nature methods,* vol. 10, p. 1177, 2013.

[31] M. Carrara, J. Lum, F. Cordero, M. Beccuti, M. Poidinger, S. Donatelli, R. A. Calogero and F. Zolezzi, "Alternative splicing detection workflow needs a careful combination of sample prep and bioinformatics analysis," *BMC bioinformatics,* vol. 16, p. S2, 2015.

[32] J. K. Pickrell, A. A. Pai, Y. Gilad and J. K. Pritchard, "Noisy splicing drives mRNA isoform diversity in human cells," *PLoS genetics,* vol. 6, p. e1001236, 2010.

[33] R. Elkon, A. P. Ugalde and R. Agami, "Alternative cleavage and polyadenylation: extent, regulation and function," *Nature Reviews Genetics,* vol. 14, p. 496, 2013.

[34] D. C. Di Giammartino, K. Nishida and J. L. Manley, "Mechanisms and consequences of alternative polyadenylation," *Molecular cell,* vol. 43, pp. 853-866, 2011.

[35] Y.-N. Kang, D.-P. Lai, H. S. Ooi, T.-t. Shen, Y. Kou, J. Tian, D. M. Czajkowsky, Z. Shao and X. Zhao, "Genome-wide profiling of untranslated regions by paired-end ditag sequencing reveals unexpected transcriptome complexity in yeast," *Molecular genetics and genomics,* vol. 290, pp. 217-224, 2015.

[36] A. Derti, P. Garrett-Engele, K. D. MacIsaac, R. C. Stevens, S. Sriram, R. Chen, C. A. Rohl, J. M. Johnson and T. Babak, "A quantitative atlas of polyadenylation in five mammals," *Genome research,* pp. gr--132563, 2012.

[37] J. Behr, A. Kahles, Y. Zhong, V. T. Sreedharan, P. Drewe and G. Rätsch, "MITIE: Simultaneous RNA-Seq-based transcript identification and quantification in multiple samples," *Bioinformatics,* vol. 29, pp. 2529-2538, 2013.

[38] F. Hormozdiari, I. Hajirasouliha, A. McPherson, E. E. Eichler and S. C. Sahinalp, "Simultaneous structural variation discovery among multiple paired-end sequenced genomes," *Genome research,* 2011.

[39] Z. Wang, M. Gerstein and M. Snyder, "RNA-Seq: a revolutionary tool for transcriptomics," *Nature reviews genetics,* vol. 10, p. 57, 2009.

[40] Y. S. Niknafs, B. Pandian, H. K. Iyer, A. M. Chinnaiyan and M. K. Iyer, "TACO produces robust multisample transcriptome assemblies from RNA-seq," *Nature methods,* vol. 14, p. 68, 2016.

[41] A. I. Tomescu, A. Kuosmanen, R. Rizzi and V. Mäkinen, "A novel min-cost flow method for estimating transcript expression with RNA-Seq," in *BMC bioinformatics*, 2013.

[42] S. Canzar, S. Andreotti, D. Weese, K. Reinert and G. W. Klau, "CIDANE: comprehensive isoform discovery and abundance estimation," *Genome biology,* vol. 17, p. 16, 2016.

[43] E. Bernard, L. Jacob, J. Mairal and J.-P. Vert, "Efficient RNA isoform identification and quantification from RNA-Seq data with network flows," *Bioinformatics,* vol. 30, pp. 2447-2455, 2014.

[44] L. Song, S. Sabunciyan and L. Florea, "CLASS2: accurate and efficient splice variant annotation from RNA-seq reads," *Nucleic acids research,* vol. 44, pp. e98--e98, 2016.

[45] M. Pertea, G. M. Pertea, C. M. Antonescu, T.-C. Chang, J. T. Mendell and S. L. Salzberg, "StringTie enables improved reconstruction of a transcriptome from RNA-seq reads," *Nature biotechnology,* vol. 33, p. 290, 2015.

[46] M. Shao and C. Kingsford, "Accurate assembly of transcripts through phase-preserving graph decomposition," *Nature biotechnology,* vol. 35, p. 1167, 2017.

[47] J. Liu, T. Yu, T. Jiang and G. Li, "TransComb: genome-guided transcriptome assembly via combing junctions in splicing graphs," *Genome biology,* vol. 17, p. 213, 2016.

[48] A. C. Frazee, A. E. Jaffe, B. Langmead and J. T. Leek, "Polyester: simulating RNA-seq datasets with differential transcript expression," *Bioinformatics,* vol. 31, pp. 2778-2784, 2015.

[49] P. K. Srivastava, M. Bagnati, A. Delahaye-Duriez, J.-H. Ko, M. Rotival, S. R. Langley, K. Shkura, M. Mazzuferi, B. Danis, J. Eyll and others, "Genome-wide analysis of differential RNA editing in epilepsy," *Genome research,* vol. 27, pp. 440-450, 2017.

[50] T. C. Glenn, "Field guide to next-generation DNA sequencers," *Molecular Ecology Resources,* vol. 11, pp. 759-769, 2011.

[51] E. C. Hayden, "Is the $1,000 genome for real?," *Nature News,* 2014.

[52] P. A. Pevzner, H. Tang and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proceedings of the National Academy of Sciences,* vol. 98, pp. 9748-9753, 2001.

[53] M. Chaisson, P. Pevzner and H. Tang, "Fragment assembly with short reads," *Bioinformatics,* vol. 20, pp. 2067-2074, 2004.

[54] J. Schröder, H. Schröder, S. J. Puglisi, R. Sinha and B. Schmidt, "SHREC: a short-read error correction method," *Bioinformatics,* vol. 25, pp. 2157-2163, 2009.

[55] L. Ilie, F. Fazayeli and S. Ilie, "HiTEC: accurate error correction in high-throughput sequencing data," *Bioinformatics,* vol. 27, pp. 295-302, 2011.

[56] L. Salmela and J. Schröder, "Correcting errors in short reads by multiple alignments," *Bioinformatics,* vol. 27, pp. 1455-1461, 2011.

[57] W.-C. Kao, A. H. Chan and Y. S. Song, "ECHO: a reference-free short-read error correction algorithm," *Genome research,* vol. 21, pp. 1181-1192, 2011.

[58] X. Yang, K. S. Dorman and S. Aluru, "Reptile: representative tiling for short read error correction," *Bioinformatics,* vol. 26, pp. 2526-2533, 2010.

[59] P. Medvedev, E. Scott, B. Kakaradov and P. Pevzner, "Error correction of high-throughput sequencing datasets with non-uniform coverage," *Bioinformatics,* vol. 27, pp. i137--i141, 2011.

[60] D. R. Kelley, M. C. Schatz, S. L. Salzberg and others, "Quake: quality-aware detection and correction of sequencing errors," *Genome Biol,* vol. 11, p. R116, 2010.

[61] G. Marçais and C. Kingsford, "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers," *Bioinformatics,* vol. 27, pp. 764-770, 2011.

[62] H. Shi, B. Schmidt, W. Liu and W. Müller-Wittig, "A parallel algorithm for error correction in high-throughput short-read data on CUDA-enabled graphics hardware," *Journal of Computational Biology,* vol. 17, pp. 603-615, 2010.

[63] Y. Liu, J. Schröder and B. Schmidt, "Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data," *Bioinformatics,* vol. 29, pp. 308-315, 2013.

[64] Y. Heo, X.-L. Wu, D. Chen, J. Ma and W.-M. Hwu, "BLESS: Bloom-filter-based Error Correction Solution for High-throughput Sequencing Reads," *Bioinformatics,* p. btu030, 2014.

[65] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM,* vol. 13, pp. 422-426, 1970.

[66] S. Tarkoma, C. E. Rothenberg and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *Communications Surveys & Tutorials, IEEE,* vol. 14, pp. 131-155, 2012.

[67] J. Pell, A. Hintze, R. Canino-Koning, A. Howe, J. M. Tiedje and C. T. Brown, "Scaling metagenome sequence assembly with probabilistic de Bruijn graphs," *Proceedings of the National Academy of Sciences,* vol. 109, pp. 13272-13277, 2012.

[68] D. C. Jones, W. L. Ruzzo, X. Peng and M. G. Katze, "Compression of next-generation sequencing reads aided by highly efficient de novo assembly," *Nucleic acids research,* vol. 40, pp. e171--e171, 2012.

[69] P. Melsted and J. K. Pritchard, "Efficient counting of k-mers in DNA sequences using a bloom filter," *BMC bioinformatics,* vol. 12, p. 333, 2011.

[70] F. Putze, P. Sanders and J. Singler, "Cache-, Hash-, and Space-efficient Bloom Filters," *J. Exp. Algorithmics,* vol. 14, pp. 4:4.4--4:4.18, 1 2010.

[71] P. Melsted and B. V. Halldórsson, "KmerStream: Streaming algorithms for k-mer abundance estimation," *bioRxiv,* 2014.

[72] R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, G. He, Y. Chen, Q. Pan, Y. Liu and others, "SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler," *Gigascience,* vol. 1, p. 18, 2012.

[73] M. Holtgrewe, "Mason--a read simulator for second generation sequencing data," *Technical Report FU Berlin,* 2010.

[74] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature methods,* vol. 9, pp. 357-359, 2012.

[75] A. Gurevich, V. Saveliev, N. Vyahhi and G. Tesler, "QUAST: quality assessment tool for genome assemblies," *Bioinformatics,* vol. 29, pp. 1072-1075, 2013.

[76] D. R. Zerbino and E. Birney, "Velvet: algorithms for de novo short read assembly using de Bruijn graphs," *Genome research,* vol. 18, pp. 821-829, 2008.

[77] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts and others, "GAGE: A critical evaluation of genome assemblies and assembly algorithms," *Genome research,* vol. 22, pp. 557-567, 2012.

[78] L. Fan, P. Cao, J. Almeida and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking (TON),* vol. 8, pp. 281-293, 2000.

[79] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh and G. Varghese, "An improved construction for counting bloom filters," in *Algorithms--ESA 2006*, Springer, 2006, pp. 684-695.

[80] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms,* vol. 55, pp. 58-75, 2005.

[81] 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," *arXiv preprint arXiv:1309.2975,* 2013.

[82] K. Nakamura, T. Oshima, T. Morimoto, S. Ikeda, H. Yoshikawa, Y. Shiwa, S. Ishikawa, M. C. Linak, A. Hirai, H. Takahashi and others, "Sequence-specific error profile of Illumina sequencers," *Nucleic acids research,* p. gkr344, 2011.

[83] R. Chikhi and P. Medvedev, "Informed and automated k-mer size selection for genome assembly," *Bioinformatics,* vol. 30, pp. 31-37, 2014.

[84] X. Yang, S. P. Chockalingam and S. Aluru, "A survey of error-correction methods for next-generation sequencing," *Briefings in Bioinformatics,* vol. 14, pp. 56-66, 2013.

[85] H. Li, "BFC: correcting Illumina sequencing errors," *Bioinformatics,* 2015.

[86] L. Song, L. Florea and B. Langmead, "Lighter: fast and memory-efficient sequencing error correction without counting," *Genome Biology,* vol. 15, p. 509, 2014.

[87] L. Salmela, "Correction of sequencing errors in a mixed set of reads," *Bioinformatics,* vol. 26, pp. 1284-1290, 2010.

[88] H.-S. Le, M. H. Schulz, B. M. McCauley, V. F. Hinman and Z. Bar-Joseph, "Probabilistic error correction for RNA sequencing," *Nucleic Acids Research,* vol. 41, p. e109, 2013.

[89] M. D. MacManes, "Optimizing error correction of RNAseq reads," *bioRxiv,* vol. doi:10.1101/020123, 2015.

[90] M. H. Schulz, D. R. Zerbino, M. Vingron and E. Birney, "Oases: robust de novo RNA-seq assembly across the dynamic range of expression levels," *Bioinformatics,* vol. 28, pp. 1086-1092, 2012.

[91] B. Walenz and L. Florea, "Sim4db and Leaff: utilities for fast batch spliced alignment and sequence indexing," *Bioinformatics,* vol. 27, pp. 1869-1870, 2011.

[92] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. K. Pham, A. D. Prjibelski, A. Pyshkin, A. Sirotkin, N. Vyahhi, G. Tesler, M. A. Alekseyev and P. A. Pevzner, "SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing.," *Journal of Computational Biology,* vol. 19, pp. 455-477, 2012.

[93] S. I. Nikolenko, A. Korobeynikov and M. A. Alekseyev, "BayesHammer: Bayesian clustering for error correction in single-cell sequencing.," *BMC Genomics,* vol. 14, p. S7, 2013.

[94] T. B. K. Reddy, A. D. Thomas, D. Stamatis, J. Bertsch, M. Isbandi, J. Jansson, J. Mallajosyula, I. Pagani, E. A. Lobos and N. C. Kyrpides, "The Genomes OnLine Database (GOLD) v. 5: a metadata management system based on a four level (meta) genome project classification," *Nucleic acids research,* vol. 43, pp. D1099--D1106, 2014.

[95] Arabidopsis Genome Initiative and others, "Analysis of the genome sequence of the flowering plant Arabidopsis thaliana," *nature,* vol. 408, p. 796, 2000.

[96] International Human Genome Sequencing Consortium and others, "Initial sequencing and analysis of the human genome," *Nature,* vol. 409, p. 860, 2001.

[97] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt and others, "The sequence of the human genome," *science,* vol. 291, pp. 1304-1351, 2001.

[98] Mouse Genome Sequencing Consortium and others, "Initial sequencing and comparative analysis of the mouse genome," *Nature,* vol. 420, p. 520, 2002.

[99] Rat Genome Sequencing Project Consortium and others, "Genome sequence of the Brown Norway rat yields insights into mammalian evolution," *Nature,* vol. 428, p. 493, 2004.

[100] International Human Genome Sequencing Consortium and others, "Finishing the euchromatic sequence of the human genome," *Nature,* vol. 431, p. 931, 2004.

[101] G. A. Tuskan, S. Difazio, S. Jansson, J. Bohlmann, I. Grigoriev, U. Hellsten, N. Putnam, S. Ralph, S. Rombauts, A. Salamov and others, "The genome of black cottonwood, Populus trichocarpa (Torr. & Gray)," *science,* vol. 313, pp. 1596-1604, 2006.

[102] M. Hunt, C. Newbold, M. Berriman and T. D. Otto, "A comprehensive evaluation of assembly scaffolding tools," *Genome biology,* vol. 15, p. R42, 2014.

[103] K. Wang, D. Singh, Z. Zeng, S. J. Coleman, Y. Huang, G. L. Savich, X. He, P. Mieczkowski, S. A. Grimm, C. M. Perou and others, "MapSplice: accurate mapping of RNA-seq reads for splice junction discovery," *Nucleic acids research,* vol. 38, pp. e178--e178, 2010.

[104] C. Trapnell, A. Roberts, L. Goff, G. Pertea, D. Kim, D. R. Kelley, H. Pimentel, S. L. Salzberg, J. L. Rinn and L. Pachter, "Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks," *Nature protocols,* vol. 7, p. 562, 2012.

[105] L. Florea, A. Souvorov, T. S. Kalbfleisch and S. L. Salzberg, "Genome assembly has a major impact on gene content: a comparison of annotation in two Bos taurus assemblies," *PLoS One,* vol. 6, p. e21400, 2011.

[106] R. A. Dalloul, J. A. Long, A. V. Zimin, L. Aslam, K. Beal, L. A. Blomberg, P. Bouffard, D. W. Burt, O. Crasta, R. P. M. A. Crooijmans and others, "Multi-platform next-generation sequencing of the domestic turkey (Meleagris gallopavo): genome assembly and analysis," *PLoS biology,* vol. 8, p. e1000475, 2010.

[107] V. Shulaev, D. J. Sargent, R. N. Crowhurst, T. C. Mockler, O. Folkerts, A. L. Delcher, P. Jaiswal, K. Mockaitis, A. Liston, S. P. Mane and others, "The genome of woodland strawberry (Fragaria vesca)," *Nature genetics,* vol. 43, p. 109, 2011.

[108] J. L. Wegrzyn, J. D. Liechty, K. A. Stevens, L.-S. Wu, C. A. Loopstra, H. A. Vasquez-Gross, W. M. Dougherty, B. Y. Lin, J. J. Zieve, P. J. Martínez-García and others, "Unique features of the loblolly pine (Pinus taeda L.) megagenome revealed through sequence annotation," *Genetics,* vol. 196, pp. 891-909, 2014.

[109] M. Pop, D. S. Kosack and S. L. Salzberg, "Hierarchical scaffolding with Bambus," *Genome research,* vol. 14, pp. 149-159, 2004.

[110] A. Dayarian, T. P. Michael and A. M. Sengupta, "SOPRA: Scaffolding algorithm for paired reads via statistical optimization," *BMC bioinformatics,* vol. 11, p. 345, 2010.

[111] M. Boetzer, C. V. Henkel, H. J. Jansen, D. Butler and W. Pirovano, "Scaffolding pre-assembled contigs using SSPACE," *Bioinformatics,* vol. 27, pp. 578-579, 2010.

[112] S. Gao, N. Nagarajan and W.-K. Sung, "Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences," in *International Conference on Research in Computational Molecular Biology*, 2011.

[113] N. Donmez and M. Brudno, "SCARPA: scaffolding reads with practical algorithms," *Bioinformatics,* vol. 29, pp. 428-434, 2012.

[114] W. Xue, J.-T. Li, Y.-P. Zhu, G.-Y. Hou, X.-F. Kong, Y.-Y. Kuang and X.-W. Sun, "L_RNA_scaffolder: scaffolding genomes with transcripts," *BMC genomics,* vol. 14, p. 604, 2013.

[115] B. J. Haas, A. Papanicolaou, M. Yassour, M. Grabherr, P. D. Blood, J. Bowden, M. B. Couger, D. Eccles, B. Li, M. Lieber and others, "De novo transcript sequence reconstruction from RNA-seq using the Trinity platform for reference generation and analysis," *Nature protocols,* vol. 8, p. 1494, 2013.

[116] S. V. Zhang, L. Zhuo and M. W. Hahn, "AGOUTI: improving genome assembly and annotation using transcriptome data," *GigaScience,* vol. 5, p. 31, 2016.

[117] D. H. Huson, K. Reinert and E. W. Myers, "The greedy path-merging algorithm for contig scaffolding," *Journal of the ACM (JACM),* vol. 49, pp. 603-615, 2002.

[118] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, Introduction to algorithms, MIT press, 2009.

[119] I. Verde, A. G. Abbott, S. Scalabrin, S. Jung, S. Shu, F. Marroni, T. Zhebentyayeva, M. T. Dettori, J. Grimwood, F. Cattonaro and others, "The high-quality draft genome of peach (Prunus persica) identifies unique patterns of genetic diversity, domestication and genome evolution," *Nature genetics,* vol. 45, p. 487, 2013.

[120] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows--Wheeler transform," *bioinformatics,* vol. 25, pp. 1754-1760, 2009.

[121] D. Chagné, R. N. Crowhurst, M. Pindo, A. Thrimawithana, C. Deng, H. Ireland, M. Fiers, H. Dzierzon, A. Cestaro, P. Fontana and others, "The draft genome sequence of European pear (Pyrus communis L.'Bartlett')," *PloS one,* vol. 9, p. e92644, 2014.

[122] 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, pp. 3389-3402, 1997.

[123] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu and S. L. Salzberg, "Versatile and open software for comparing large genomes," *Genome biology,* vol. 5, p. R12, 2004.

[124] H. Hirakawa, K. Shirasawa, S. Kosugi, K. Tashiro, S. Nakayama, M. Yamada, M. Kohara, A. Watanabe, Y. Kishida, T. Fujishiro and others, "Dissection of the octoploid strawberry

genome by deep sequencing of the genomes of Fragaria species," *DNA research,* vol. 21, pp. 169-181, 2014.

[125] L. Song and L. Florea, "Software and exemplar data for Rcorrector," *GigaScience Database http://dx.doi.org/10.5524/100171,* 2015.

[126] R. S. Roy, D. Bhattacharya and A. Schliep, "Turtle: Identifying frequent k-mers with cache-efficient algorithms," *arXiv preprint arXiv:1305.1861,* 2013.

[127] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *arXiv preprint arXiv:1303.3997,* 2013.

[128] D. Kim, G. Pertea, C. Trapnell, H. Pimentel, R. Kelley and S. L. Salzberg, "Accurate alignment of transcriptomes in the presence of insertions, deletions and gene fusions," *Genome Biology,* vol. 14, p. R36, 2013.

[129] S. Istrail, G. G. Sutton, L. Florea, A. L. Halpern, C. M. Mobarry, R. Lippert, B. Walenz, H. Shatkay, I. Dew, J. R. Miller and others, "Whole-genome shotgun assembly and comparison of human genome assemblies," *Proceedings of the National Academy of Sciences,* vol. 101, pp. 1916-1921, 2004.

[130] A. Doring, D. Weese, T. Rausch and K. Reinert, "SeqAn: An efficient, generic C++ library for sequence analysis," *BMC Bioinformatics,* vol. 9, p. 11, 2008.

# Curriculum Vitae

Li Song was born in Chengdu, Sichuan, China in 1987. He received a Bachelor of Engineering degree from the Computer Science and Technology Department of Tongji University in 2009. Between 2009-2012, he attended the Computer Science Department at the Michigan Technological University, where he worked on high performance computing advised by Prof. Steven Seidel. In 2012, he started his research in computational biology at the Department of Computer Science at the Johns Hopkins University, where he designed algorithms for next generation sequencing data analysis under the advisorship of Prof. Liliana Florea. Li Song obtained a Master of Science degree in Computer Science from the Michigan Technological University in 2011, a Master of Science degree in Applied Mathematics and Statistics from the Johns Hopkins University in 2017, and a PhD degree in Computer Science from the Johns Hopkins University in 2018.