UNIVERSITY OF CALIFORNIA
RIVERSIDE

Algorithms for Reference Assisted Genome and Transcriptome Assemblies

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Ergude Bao

June 2014

Dissertation Committee:

Professor Thomas Girke, Chairperson
Professor Tao Jiang
Professor Stefano Lonardi
Professor Marek Chrobak

The Dissertation of Ergude Bao is approved:

Chairperson

University of California, Riverside

Acknowledgments

First of all, I thank my advisors Thomas Girke and Tao Jiang, my previous advisor Weisheng Li at the Beijing Jiaotong University, and my previous group leader Dongrui Fan at the Institute of Computing Technology, Chinese Academy of Sciences. I have really learned a lot about how to perform good research with your tireless help over the years.

Second, I thank the School of Software Engineering at the Beijing Jiaotong University, who gave me the opportunity to start my academy career as an associate professor. I will prove that we both made a good choice.

Finally, I thank all my friends around the world for their continuous support. I thank my labmates who always helped me and gave me a lot of suggestions during my research. I thank the people from the churches in Riverside for all their encouragement and help, especially when I just arrived at Riverside in 2009. I thank everyone in my family who keep taking care of every aspect of my life. I lover you all.

To my mom Min Wang and in memory of my grandma Chunfang Li.

ABSTRACT OF THE DISSERTATION

Algorithms for Reference Assisted Genome and Transcriptome Assemblies

by

Ergude Bao

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2014
Professor Thomas Girke, Chairperson

De novo genome and transcriptome assemblies of next generation sequences (NGS) are important for many genomics applications of unsequenced organisms. Both assembly types present many challenges owing to (i) the large amount of data to process, (ii) sequencing errors and (iii) the complexity of transcriptomes and genomes. The latter is a result of alternative splice events, variable and incomplete representations of transcripts in RNA-Seq libraries, while repetitive regions in genomes complicate their assembly. Usually, *de novo* assemblies result in thousands of short and incomplete transcripts (transfrags) or genomic sequences (contigs or scaffolds) while requiring a large amount of processing time and memory. However, with decreasing NGS costs reference genomes of many species have become available recently that can be used to guide and improve *de novo* assemblies. Here, we introduce two reference assisted algorithms BRANCH and AlignGraph. BRANCH improves transcriptome assemblies guided by genomic contigs from the same species or reference genes from a closely related species. AlignGraph improves genome assemblies with help provided by a closely related reference genome. In addition, we introduce the short read clustering algorithm SEED that is useful as a preprocessing tool in *de novo*

assemblies by reducing their time and memory requirements.

SEED joins sequences into clusters that can differ by up to three mismatches and three overhanging residues from their virtual center. It is based on a modified spaced seed method, called block spaced seeds. Its clustering component operates on the hash tables by first identifying virtual center sequences and then finding all their neighboring sequences that meet the similarity parameters. SEED can cluster 100 million short read sequences in <4 h with a linear time and memory performance. When using SEED as a preprocessing tool on genome/transcriptome assembly data, it was able to reduce the time and memory requirements of the Velvet/Oasis assembler, for the datasets used in this study, by 60-85% and 21-41%, respectively. In addition, the assemblies contained longer contigs than non-preprocessed data as indicated by 12-27% larger N50 values. Compared to other clustering tools, SEED showed the best performance in generating clusters of NGS data similar to true cluster results with a 2- to 10-fold better time performance. While most of SEED's utilities fall into the preprocessing area of NGS data, our tests also demonstrate its efficiency as stand-alone tool for discovering clusters of small RNA sequences in NGS data from unsequenced organisms.

BRANCH's input includes assembled RNA reads, genomic sequences (e.g. contigs) and the RNA reads themselves. It uses a customized version of BLAT to align the transfrags and RNA reads to the genomic sequences. After identifying exons from the alignments, it defines a directed acyclic graph and maps the transfrags to paths on the graph. It then joins and extends the transfrags by applying an algorithm that solves a combinatorial optimization problem, called the Minimum weight Minimum Path Cover with given Paths.

In performance tests on real data from *Caenorhabditis elegans* and *Saccharomyces cerevisiae*, assisted by genomic contigs from the same species, BRANCH improved the sensitivity and precision of transfrags generated by Velvet/Oases or Trinity by 5.1-56.7% and 0.3-10.5%, respectively. These improvements added 3.8-74.1% complete transcripts and 3.8-8.3% proteins to the initial assembly. Similar improvements were achieved when guiding the BRANCH processing of a transcriptome assembly from a more complex organism (mouse) with genomic sequences from a related species (rat).

AlignGraph is an algorithm for extending and joining *de novo* assembled contigs or scaffolds guided by closely related reference genomes. It aligns paired-end (PE) reads and pre-assembled contigs or scaffolds to a close reference. From the obtained alignments, it builds a novel data structure, called the *paired-end multi-positional de Bruijn graph*. The incorporated positional information from the alignments and PE reads allows us to extend the initial assemblies, while avoiding incorrect extensions and early terminations. In our performance tests, AlignGraph was able to substantially improve the contigs and scaffolds from several assemblers. For instance, 28.7-62.3% of the contigs of *Arabidopsis thaliana* and human could be extended, resulting in improvements of common assembly metrics, such as an increase of the N50 of the extendable contigs by 89.9-94.5% and 80.3-165.8%, respectively. In another test, AlignGraph was able to improve the assembly of a published genome (*Arabidopsis* strain Landsberg) by increasing the N50 of its extendable scaffolds by 86.6%. These results demonstrate AlignGraph's efficiency in improving genome assemblies by taking advantage of closely related references.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Next Generation Sequencing

Next Generation Sequencing (NGS) emerged in 2005 and is improving with a very fast pace. Widely used NGS platforms include Illumina, Roche 454 and SOLiD. Although each platform has its own distinct characteristics, the basic work flows are similar (Shendure and Ji, 2008). The first step of the NGS method is to break DNA into shorter template fragments. After ligating adaptors, the fragments are immobilized on a flow cell and then amplified via a bridge *Polymerase Chain Reaction* (PCR) step to so called clusters. All clusters on a flow cell are sequenced simultaneously using a reversible fluorescence terminator approach. The NGS technology can also sequence transcriptomes after converting RNA to DNA via a reverse transcriptase. However, it is usually harder to generate high-quality RNA reads than DNA reads because of severe abundance differences and secondary structures in mRNAs. Compared with Sanger sequencing, NGS is much faster and less expensive, but the

reads are often shorter and of lower quality. Assembling DNA or RNA reads to genomes or transcriptomes, respectively, is one of the most challenging problems in bioinformatics. The following introduces the algorithms commonly used for genome and transcriptome assemblies. Subsequently, unsolved problems and opportunities of the assembly field are discussed.

1.2 Genome Assembly Algorithms

The most challenging aspects of genome assemblies are limited read quality and repetitive regions in genome sequences. Read quality is affected by sequencing errors and lowly sequenced genomic regions (or sequencing gaps). Low read quality can result in false positive and incomplete contigs, respectively. Repetitive sequence regions can cause ambiguity in the assembly limiting contig lengths. Most genome assembly algorithms are *de novo* methods. A much smaller number of algorithms incorporates information from genomes of closely related species. The *de novo* algorithms can be further classified into three major types of algorithms: overlap-then-extend, string graph and de Bruijn graph algorithms. The following introduces these different types of algorithms in more detail.

1.2.1 *De Novo* Genome Assembly Algorithms

(A) Overlap-then-extend Algorithm. The assemblers using the *overlap-then-extend* approach were developed in the early stages of the NGS technologies. Their basic idea is to iteratively extend a contig if the coverage and the number of overlapping bases are both above a minimum threshold. This operation is repeated until no more reads can be joined.

The algorithm is relatively straightforward to implement, but it has two major drawbacks: (i) extensions are often terminated when the process transitions from a repetitive to a non-repetitive region, and (ii) the time performance of this approach is not sufficient for processing tens of millions of reads even when using relatively efficient suffix tree based read indexing methods. The first assembler using this approach was SSAKE (Warren *et al.*, 2007a). VCAKE is an improved variant allowing mismatches in the extension steps to account for base call errors (Jeck *et al.*, 2007), while SHARCGS uses several read quality filtering steps to improve the precision of the contigs (Dohm *et al.*, 2007).

(B) String Graph-based Algorithm. Kececioglu and Myers (1995) were the first to propose a string graph-based algorithm for Sanger read assembly, and Myers (2005) applied the idea to NGS data. A string graph is a directed graph where each vertex represents a read, and one vertex u is connected to another vertex v if the suffix of at least x bases of u is the same as the prefix of v. To construct a string graph, an important pre-process is to find all the overlaps between each read pair. After the initial graph layout is generated, the graph is simplified by reducing transitive edges, and then the copy number of each edge is computed to infer how many times each edge should be traversed. Finally, the algorithm finds the minimum cost flow from the copy number weighted graph and the obtained flow corresponds to the target genome. The advantage of the algorithm is its memory efficiency. However, a major limitation is its compute time for finding the overlaps of each read pair even with recent improvements from Rasmussen *et al.* (2006) or the adoption of the FM-index to this problem (Simpson and Durbin, 2010; Ferragina and Manzini, 2000; Li and Durbin, 2009a). Edena and SGA are two assembly software tools based on this approach (Hernandez *et al.*,

2008; Simpson and Durbin, 2012).

(C) De Bruijn Graph-based Algorithm. Initially, Idury and Waterman (1995) introduced the de Bruijn graph algorithm for Sanger read assembly. Pevzner *et al.* (2001) extended this idea by incorporating error correction techniques and paired-end read support, and Chaisson and Pevzner (2008) applied it to NGS data. A de Bruijn graph is a directed graph. Two connected vertices u and v represent $k+1$ bases where u represents the first k bases and v the second k bases (called *k-mer*). To construct a de Bruijn graph, $l-k+1$ connected vertices are constructed from each read of length l and two vertices from different reads are joined if they have the same k-mers. Ideally, a traversal of the de Bruijn graph would cover all the bases of a genome in sequential order, and thus reconstruct its entire sequence. In comparison to the overlap graph-based NGS assembly algorithms, the de Bruijn graph approach has two major advantages making it the preferred method for most NGS assembly tools. First, it avoids the time-consuming computation step to find overlaps between each read pair. Second, it can easily process reads of variable lengths. A disadvantage is its relatively large memory footprint requiring high performance compute environments for assembling larger genomes.

Software examples using the de Bruijn graph include Velvet (Zerbino and Birney, 2008a), ABySS (Simpson *et al.*, 2009), ALLPATHS (Butler *et al.*, 2008; Gnerre *et al.*, 2011), SOAPdenovo (Li *et al.*, 2010; Luo *et al.*, 2012), MaSuRCA (Zimin *et al.*, 2013), CABOG (Miller *et al.*, 2008) and Euler-SR (Chaisson and Pevzner, 2008). Euler-SR was the first release of this type of assemblers, while Velvet and ALLPATHS contain several modifications and improvements (Miller *et al.*, 2010). Each vertex of the de Bruijn graph built by Velvet

is not a single k-mer, but a set of continuous k-mers. This design helps preserving k-mer connections contained in the reads. ALLPATHS-LG, an improved version of ALLPATHS (Gnerre *et al.*, 2011), is optimized for using two PE read sets with different insert lengths. One library with insert length $I >> 2l$ and the other with insert length $I < 2l$ where l is the read length. ALLPATHS-LG fills in the gaps between the first library's read pairs with the second library, and then assembles the extended reads. For single library assemblies, several performance comparisons among different assemblers have been published (*e.g.* Lin *et al.*, 2011; Zhang *et al.*, 2011). These publications conclude that each assembler has its own data-dependent advantages and disadvantages with respect to sensitivity, precision, run time and memory usage. However, for multiple libraries, the comparison results from the GAGE project (Salzberg *et al.*, 2012) show that ALLPATHS-LG and SOAPdenovo are currently one of the best performing genome assemblers.

More recent theoretical advances in this area include IDBA (Peng *et al.*, 2010), the paired-end de Bruijn graph (Medvedev *et al.*, 2011) and the positional de Bruijn graph (Ronen *et al.*, 2012). IDBA is an improved iterative algorithm which constructs the de Bruijn graph by iterating over all k-mer lengths. During this process the algorithm efficiently corrects errors, closes gaps in the graph and generates longer contigs. The paired-end de Bruijn graph is constructed from PE reads where each k-mer contains k bases from the left read plus k corresponding bases from the right read of a pair. With this additional information, two k-mers from repetitive regions can often be distinguished, and thus branches in the de Bruijn graph can be reduced. At this point, there is no published software available using the paired-end de Bruijn graph. The main reason for this may be its dependency on

the strand information of the reads relative to the target genome. The latter is not easily obtainable in pure *de novo* assembly applications that lack an alignment step against a reference. Currently, the positional de Bruijn graph, as implemented in the SEQuel software, is not directly used as a *de novo* assembler. Instead it is used as a downstream contig refinement method. The positional de Bruijn graph is built from the read alignments to the contigs with each k-mer containing k bases plus the corresponding mapping position of the alignment. Similar to the paired-end de Bruijn graph, branches can be reduced with the additional positional information.

1.2.2 Reference-Based Genome Assembly Algorithms

Previous work on reference-guided assemblies include the AMOScmp software (Pop *et al.*, 2004a), an add-on tool for the ARACHNE assembler (Gnerre *et al.*, 2009) and a custom workflow based on existing software tools (*e.g.* Schneeberger *et al.*, 2011). The first two have been designed primarily for Sanger read assembly, while the latter has been used for NGS assembly. The method from Gnerre *et al.* (2009) was able to substantially improve the quality of genome assemblies of complex organisms. It also has been successfully used for assembling Sanger reads from 29 mammalian genomes (Lindblad-Toh *et al.*, 2011). This heuristic algorithm aligns both pre-assembled *de novo* contigs and Sanger reads to a reference genome, extends and joins the contigs or scaffolds with the reads. The pipeline approach from Schneeberger *et al.* (2011) was developed to assemble four strains of *Arabidopsis thaliana* from paired-end Illumina reads using a local assembly approach of reads aligning to the reference and those that could not be aligned with a combination of *de novo*

assemblers. The merged and error-corrected contigs were then further assembled into supercontigs using a scaffolding software. Currently, the authors provide no software to easily rerun their assembly pipeline on other data sets.

1.3 Transcriptome Assembly Algorithms

A major challenge in transcriptome assemblies are splice variants containing different combinations of exons encoded by each gene in a genome. Thus, an assembly algorithm has to find the correct combination of exons contained in each transcript. Similar to genome assemblies, transcriptome assemblies are affected by poor read quality, while uneven read coverage for different transcripts makes it hard to use the coverage information to correct sequencing errors. There are two types of transcriptome assembly algorithms: *de novo* algorithms and reference-based algorithms. The two types of algorithms differ in many aspects. The de Bruijn graph is the major choice for the *de novo* algorithms while the string graph is more convenient for reference-based algorithms since the overlaps between each read pair can be easily obtained by aligning the reads to the reference. The following discusses the two types of algorithms. It is important to note that the reference genome for the reference-based genome assembly approach is from a closely related species, whereas reference assisted transcriptome assemblies utilize the genome from the same species.

1.3.1 *De Novo* Transcriptome Assembly Algorithms

Most *de novo* transcriptome assembly algorithms are based on the de Bruijn graph. Unlike genome assembly algorithms, these algorithms pre-assemble the reads into intermediate

contigs, cluster the contigs and then build a de Bruijn graph for each cluster. The contigs in the same cluster are expected from the same gene sharing bases of sufficient length, so the de Bruijn graph built from a cluster is much simpler than in genome assemblies. The graph contains branches, but unlike in genome assemblies, most of the branches come from splice variant events. Both paired-end read information and coverage difference among paths can be used to resolve the branches to generate transfrags. Representative assemblers in this category include Trinity (Grabherr *et al.*, 2011), Oases (Schulz *et al.*, 2012), Trans-ABySS (Robertson *et al.*, 2010), SOAPdenovo-Trans (Xie *et al.*, 2013) and T-IDBA (Peng *et al.*, 2011). While all of them share the same algorithmic foundation, Trans-ABySS and Oases assemble transfrags with different sizes of k-mers and then merge the different transfrag sets. This multiple k-mer approach can improve the sensitivity of assemblies.

1.3.2 Reference-Based Transcriptome Assembly Algorithms

The algorithms in this category mainly include Cufflinks (Trapnell *et al.*, 2010), Scripture (Guttman *et al.*, 2010), IsoInfer (Feng *et al.*, 2011) and IsoLasso (Li *et al.*, 2011). Cufflinks first builds a string graph (referred to as overlap graph) by aligning the reads to the reference genome and then finding the overlaps among the reads. It then weights the edges of the graph with the aligned reads and finds the minimum weight minimum path cover representing the transcripts. Scripture builds a connectivity graph where each node represents a base and each directed edge is placed from one node to another if the corresponding bases are covered by two continuous read bases. It then finds and outputs all the possible paths from the connectivity graph. IsoInfer subdivides the discovered exons into segments

and minimizes the observed and predicted expression levels in all the segments with a least squares objective function. IsoLasso is an updated version of IsoInfer by adding an L1 norm penalty term to the least squares objective function in order to achieve better assembly results. In summary, Cufflinks seeks the minimum set of transcripts corresponding to the reads. IsoInfer minimizes the discrepancy between the observed and predicted expression levels, while IsoLasso minimizes both. Scripture lacks an optimization step which negatively impacts its performance.

1.3.3 Comparison between *De Novo* and Reference-Based Transcriptome Assembly Algorithms

The reference-based algorithms are generally more sensitive than the *de novo* algorithms while the *de novo* algorithms do not require a reference genome. The reference-based algorithms are affected by errors in the reference genome and false read alignments, while *de novo* algorithms require much higher sequencing depth and memory resources.

1.4 Problems and Opportunities

1.4.1 Completeness of Genome and Transcriptome Assemblies

As discussed above, sequencing errors and sequence gaps affect both genome and transcriptome assemblies, while transcriptome assemblers cannot easily take advantage of the read coverage information to correct sequencing errors due to abundance differences of transcripts. Besides this, genome assemblies are challenging mainly due to repetitive regions in genomes, and the main challenge in transcriptome assemblies are splicing events leading

to populations of highly similar transcript variants. As a result, even the most advanced genome and transcriptome assemblers usually fail to assemble complete genomes or transcriptomes. Nevertheless, many opportunities have been evolving from NGS technologies in the assembly field. Two important observations are: (i) the cost of DNA-Seq is decreasing exponentially and (ii) the number of completely sequenced organisms is also increasing very fast. These developments lead to the following opportunities for genome and transcriptome assemblies.

(A) Opportunities for Genome Assemblies. For genome assemblies, we may refer to a closely related species which has been fully sequenced and published. Combining both *de novo* assembly and alignment-based approaches presents a powerful alternative when a closely related reference genome sequence is available, but its genetic differences relative to the target genome are too pronounced to resolve them with an alignment approach alone (Schneeberger *et al.*, 2011; Phillippy *et al.*, 2008; Schatz *et al.*, 2013). In this case, one can first assemble the reads into contigs and then align them together with the reads to the reference. The much longer contigs facilitate the identification of complex rearrangements, while the read alignments are useful for detecting smaller variations in regions that are not covered by contigs. Due to the rapidly increasing number of reference genomes becoming available for most organism groups, this *reference-assisted assembly approach* will soon become the default option for many genome sequencing projects. Compared to *de novo* assemblies, reference-assisted assemblies have many advantages. First, the alignments of the contigs and reads against the close reference provide valuable proximity information that can be used to extend contigs with additional reads and to join contigs even if they

overlap only by a few nucleotides. Second, the proximity information in the alignments can also be used to orient and order contigs along the reference to build a scaffold map of the entire assembly. Third, the alignment map can be used to evaluate the quality of contigs and pinpoint potential mis-assemblies.

(B) Opportunities for Transcriptome Assemblies. For transcriptome assemblies, we may either refer to a closely related species or assemble the DNA reads from the same species to contigs and use the contigs to improve transcriptome assemblies. This hybrid approach of guiding transcriptome assemblies with preliminary genomic sequencing information is a practical and cost effective possibility since one can sequence nowadays a genomic sample of a 1 GB genome of interest at 20-50 coverage with the read output from only 1-2 flow cell lanes of a modern NGS instrument. Technically, the collection and sequencing of a genomic sample is also very straightforward, and stability issues or abundance variations of sequences are less a concern with genomic DNA than RNA. Alternatively, the genome contigs can be substituted by an existing genome sequence from a related species with high enough DNA sequence identity (usually $> 90 - 95\%$) to the RNA-Seq sample. This option eliminates the need for generating the genomic contig data set. The genomic sequences provide an additional backbone of evidence for improving *de novo* transcriptome assemblies by minimizing their typical errors and limitations, such as incomplete transfrags (*e.g.* missing exons), fragmented transfrags, chimeric transfrags due to low read coverage and base calling errors. When aligning the transfrags and RNA-Seq reads against given genomic contigs, one can extend and correct many of these fragmented or incomplete transfrags. For instance, two transfrags aligned next to each other on the same contig can be joined if a sufficient

number of RNA reads can be aligned to support this merge. Similarly, a transfrag can be extended if the RNA read coverage along the corresponding region of the genomic sequences indicates a truncated transcript sequence. Because genomic contigs also contain errors, it is important to allow in this process only those transfrag modifications that are supported by high-quality alignments.

1.4.2 Run Time and Memory Performance

The data volumes generated by NGS technologies have been growing at a pace that has now begun to greatly challenge the data processing and storage capacities of modern compute systems (Medini *et al.*, 2008). Only four years ago, NGS technologies like Illumina's reversible terminator method or ABI's ligation approach created approximately 1 billion bases of DNA sequence information per instrument run which has now increased to over 300 billion bases per run with even shorter turnaround times (Holt and Jones, 2008). Processing and storing the large amounts of data produced by these technologies is a major challenge for modern genome research. Thus, it is important to develop methods that can improve the efficiency of the analysis workflows for NGS data. To mention just a few, these include algorithms for processing the data more time and space efficiently (Langmead *et al.*, 2009a; Li and Durbin, 2009b; Li *et al.*, 2009) as well as data reduction approaches that aim to retain only the scientifically relevant and non-redundant information from NGS projects rather than everything (Leinonen *et al.*, 2010). For example, in genome resequencing projects one can greatly reduce the data set sizes by storing only genetic variations, while removing the bulk of the sequence information that only confirms what is already

known (Fritz *et al.*, 2011). Similarly, in quantitative NGS experiments for profiling pools of mRNAs, small RNAs or protein-DNA interactions one can convert the data to much less storage intensive tag counts at an early stage of the analysis workflow. Solutions that prevent or greatly minimize information loss are always preferred. However, with the current growth rates of NGS data many of them may soon become impractical, especially when the data sizes become the main time and financial bottleneck for conducting scientific experiments in the NGS field.

1.5 Organization of Dissertation

The proposal first introduces three algorithms designed to address the challenges discussed above. First, SEED (section 2; Bao *et al.*, 2011) improves the time and memory performance of genome and transcriptome assemblies. Second, BRANCH (section 3; Bao *et al.*, 2013) takes genomic contigs or a closely related reference genome to boost transcriptome assemblies. Third, AlignGraph (section 4; to appear) extends and joins contigs or scaffolds by reassembling them with help provided by a reference genome of a closely related species. Finally, a conclusion section proposes improvements and future advancements of these algorithms.

Chapter 2

SEED Algorithm

2.1 Introduction

This study introduces a new algorithm capable of clustering NGS sets in size ranges of several hundred million entries using a modified spaced seed method (Ma *et al.*, 2002; Lin *et al.*, 2008). This method, hereafter referred to as SEED, efficiently joins sequences into clusters with user-definable similarity parameters ranging from 0 to 3 mismatches and overhanging ends with up to 3 nucleotides in length. These mismatch features are important to make the method less sensitive to base call errors, imprecise molecular cleavage events or inaccurate adaptor trimming. The main utilities of SEED are the identification, enumeration and removal of redundant sequences in NGS data. In its current implementation, SEED is designed to function as a short read clustering tool with controllable mismatch parameters, but not as an error corrector like FreClu (Qu *et al.*, 2009). There are several practical applications of this clustering approach. First, the method can be used to reduce

the complexity in NGS data by collapsing redundant reads to a single center sequence along with its frequency information. While this data reduction step results only in a minor information loss, it can greatly improve the run time, memory requirements, and quality of genome and transcriptome assemblies. Second, it can be used to determine the sequence diversity in quantitative NGS profiling data sets, such as RNA-Seq and ChIP-Seq, by enumerating very similar reads. The resulting numbers of unique versus redundant reads can be an important parameter for identifying technical problems in these data sets (*e.g.* low reproducibility due to bias in PCR amplification steps). Third, the method can be applied to discover clusters of microRNAs (miRNAs) directly from NGS data without the requirement of mapping the reads to a reference genome which is particularly important when working with unsequenced organisms (Montgomery *et al.*, 2008; Johnson *et al.*, 2009).

While in the past decade there has been extensive research on sequence family clustering for handling data sets in the range of hundreds of thousand entries (*e.g.* Li and Godzik, 2006), there has been very limited development of methods for clustering the much larger sequence volumes from NGS experiments with hundreds of millions of entries. The short list of tools capable of clustering data sizes in the range of at least several million sequences includes UCLUST and FreClu (Edgar, 2010; Qu *et al.*, 2009). Most other clustering tools in this area are designed to solve problems related to EST analysis, such as pre-clustering of ESTs to facilitate their downstream assemblies (Rao *et al.*, 2010; Hazelhurst *et al.*, 2008; Huang and Madan, 1999; Picardi *et al.*, 2009).

In the following, we first describe the theory behind the SEED clustering algorithm as well as the design of its software implementation. We then illustrate and discuss

its time, memory and accuracy performance by using both simulated and real NGS data sets. The real data sets were specifically chosen to evaluate the algorithm's efficiency for several application areas, including complexity reduction of RNA-Seq profiling experiments in the absence of a reference, prediction of mature miRNAs, and transcriptome and genome assemblies.

2.2 Methods

2.2.1 Overview of the Algorithm

To cluster NGS by similarity, SEED indexes the reads by using the open hashing technique and a special class of spaced seeds (Lin *et al.*, 2008), called *block spaced seed*. Once the reads are stored in hash tables, SEED clusters them by first creating a *virtual center sequence* for each cluster and then finding all the reads that are within a certain similarity threshold to the center sequence. The following is a short overview of the algorithm. More details are provided in the next subsections.

A. Indexing

1. Initialize the indexing if the longest and the shortest read sequences do not differ by more than five bases in length.

2. Use the first seed in a chosen set of block spaced seeds to hash the sequences into a hash table.

3. Repeat step A.2 with each block spaced seed of the set and store their results in separate hash tables.

B. Clustering

1. Select an arbitrary sequence, identify for it all sequences within twice the mismatch threshold and compute their virtual center sequence.

2. Find for the virtual center sequence all sequences with the allowed number of overhanging bases and mismatches. Then remove these sequences from the hash tables.

3. Repeat steps B.1 to B.2 until the hash tables are empty.

2.2.2 Indexing and Hash Tables

Spaced seeds were introduced by Ma et al. (2002) as a time efficient method for sequence similarity searching. Several NGS alignment tools are based on this method. These include Eland (Cox, unpubl.), MAQ (Li *et al.*, 2008), SeqMap (Jiang and Wong, 2008) and ZOOM (Lin *et al.*, 2008). The general framework of spaced seeds can be summarized as follows. A spaced seed of length l is a binary string of l bits. When the seed is used in matching a query string of length l with another string, the bit 1 demands a match while the bit 0 tolerates a mismatch. Such a seed can also be conveniently used to index sequences of length l in hashing. For example, the spaced seed '01110' will file the sequences 'CAAAG' and 'TAAAA' into the same bucket, as well as all other five-mers with an 'AAA' in the middle. The weight w of a spaced seed is its number of 1's. It directly affects the size of the hash tables in the above indexing scheme, and thus memory usage. The parameter k is usually a predefined value, and the size of a set of spaced seeds is denoted as c. The details of designing a set of spaced seeds with full search sensitivity for given values of l, w, k will be discussed in Section 2.2.4.

The hash table data structure used in SEED is shown in Figure A-1. Each hash table corresponds to a spaced seed, and each bucket in it corresponds to a word of w bases. A bucket consists of a header and a dynamically allocated array of pointers. The header points to an array, and each pointer in the array references a sequence. During the clustering process, a tag will be assigned to the pointers where the sequences have been assigned to clusters to indicate their removal from the hash tables. In addition, there is an array of unsigned integers (not shown in Figure A-1) for storing the number of pointers in each bucket. Suppose that n is the total number of sequences. The memory usage s in bytes B on a 64-bit machine can be estimated as follows:

$$s = 3 \times 4^{w+1}c + 4nc + (\lceil \frac{l}{4} \rceil + 1)n \tag{2.1}$$

In Figure A-1, from left to right, the headers take $4^w \times c \times 8B = 2 \times 4^{w+1}cB$ memory, where $8B$ is the memory required for a pointer on a 64-bit machine in a straightforward implementation. However, integer offsets can be used instead of real pointers to reduce the memory footprint of a pointer to $4B$. The c hash tables take $n \times c \times 4B = 4ncB$ memory. The memory requirement for storing the sequences themselves is $n \times \lceil \frac{l}{4} \rceil B = n\lceil \frac{l}{4} \rceil B$ and nB for the tags. In addition, the array for storing the number of items in each bucket takes $4^w \times c \times 4B = 4^{w+1}cB$ memory. Combined together, the total memory required is $3 \times 4^{w+1}c + 4nc + \lceil \frac{l}{4} \rceil nB$. For example, if there are 1 million sequences of 36 bps, $w = 12$ and $c = 10$, then the memory requirement totals: $s = 3 \times 4^{12+1} \times 10 + 4 \times 1M \times 10 + (\lceil \frac{36}{4} \rceil + 1) \times 1MB = 1970MB$.

2.2.3 Design of Block Spaced Seed Set

While other spaced seeds methods are more common, especially in the NGS alignment field, we have chosen *block spaced seeds* for NGS clustering, because they are conceptually simple and easy to optimize.

Definition 1. *A block spaced seed is a binary string consisting of a sequence of blocks of equal length, where each block contains either all 0's or all 1's.*

The seed sets used by various short read alignment tools are usually heuristic designs. With the exception of ZOOM, they provide suboptimal solutions, but with good performance in practice. Typically, their seed sets are often the outcome of manual optimization procedures for a given read length and number of mismatches. In contrast to this, an optimal set of *block spaced seeds* for a given read length and number of mismatches can be automatically identified with Algorithm 1 (see below). Note that such an optimal set of block spaced seeds typically represents a suboptimal solution for general spaced seeds. We first state a theorem upper bounding the size of an optimal block spaced seed set. The proof of the theorem and the analysis of Algorithm 1 are both available in the Appendix section.

Theorem 1. *For any given l, w, k, there exists a set of block spaced seeds with length l and weight w that guarantees full search sensitivity with respect to k mismatches if and only if $k \leq \frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$, where $gcd(l, w)$ denotes the greatest common divisor of l and w. Moreover, for any $k \leq \frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$, $\binom{\frac{w}{gcd(l,w)}+k}{k}$ block spaced seeds of length l and weight w would suffice to guarantee full search sensitivity with respect to k mismatches.*

Algorithm 1 BestSeedSet(l, k)

$m = \infty$
for $w = 13$ **to** 11 **do**
  if $k \leq \frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$ **then**
    $c = \binom{\frac{w}{gcd(l,w)}+k}{k}$
    if $m > 2 \times 4^{w+1}c$ **then**
      $m = 2 \times 4^{w+1}c$
      $w_0 = w$
      $c_0 = c$
    end if
  end if
end for
generate block spaced seed set
return w_0 and c_0

Although it is desirable to maximize w in order to be time efficient, the memory complexity given in Equation (1) suggests that we should minimize w (and c) in order to be memory efficient. Therefore, we should seek a balance between time and space. Table A-1 shows the memory usages (headers only), seed weights and numbers of seeds required for several read lengths ranging from 25 to 35, where the seed weights and numbers of seeds for each read length are calculated using Algorithm 1 and the memory usages calculated using Equation (1) with similarity threshold $k = 3$. Clearly, if a set of block spaced seeds guarantees full sensitivity for read sequences of length l, then it also guarantees full sensitivity for sequences of length more than l. Moreover, we can always pad spaced seeds with 0's so they have the same length as the reads. Thus, for a specific pair of weight w and number c, the length l listed in the table should be regarded as the minimum read length that w and c support. Since a row with a small l, large w, small c, and small memory usage s is desirable, we choose the row with $l = 30$, $w = 12$, $c = 10$, and $s = 1.25$ GB in our experiments (where the reads are 36 bps long, up to three overhanging bases are allowed on each side and up to three mismatches are tolerated). Table A-2 lists the 10 block spaced seeds used in our experiments.

2.2.4 Clustering

The actual sequence clustering component of SEED is an iterative process consisting of three major steps. First, an arbitrary sequence x is selected and hashed using each block spaced seed to locate c buckets. The sequences in the c buckets with at most fk mismatches to the sequence x are identified by a simple Hamming distance calculation, where k is the maximum number of mismatches allowed in a cluster, and f is set to be 2 as a factor of k. The consensus of the resulting sequence set is computed to obtain a virtual center sequence. Second, the virtual center sequence is hashed using each block spaced seed, and the sequences from all the resultant buckets are retrieved. A cluster is formed to include all the sequences with $\leq k$ mismatches to the virtual center sequence. The clustered sequences are removed from the hash tables. Third, to also include sequences that largely overlap with x but with overhanging ends, the virtual center sequence is shifted (actually, rotated) to the left and to the right within the maximum allowed shift distance (predefined value from 0-3). For each shifted center sequence, all sequences in the hash tables are added to the cluster that are within k mismatches to the center and then they are also deleted from the hash tables. The above steps are repeated until all sequences have been assigned to clusters and deleted from the hash tables.

Our choice of $f = 2$ in the initial clustering (step one) is based on the following considerations. Given a cluster of sequences with $\leq k$ mismatches to its center, an arbitrarily selected sequence in the cluster has $\leq 2k$ mismatches to any sequence in the set. With this setting the method can collect all sequences belonging to a cluster even if the randomly chosen seed sequence is far away from the true center of a cluster. The final virtual center

sequence - generated from this candidate set - will then provide a reasonable approximation of the true center.

After the clustering, each sequence will be part of a cluster with one or more members. The final results are stored in two cluster result files. One tabular file lists the complete set of reads with their corresponding cluster identifiers. The second file is the clustered FASTQ file containing for each cluster only its center sequence along with the corresponding quality scores (see below).

2.2.5 Incorporating Base Calling Quality Values

NGS data contain base calling quality information usually in the form of Phred scores (Cock *et al.*, 2010). To incorporate this quality information into the clustering process, the SEED algorithm allows the user to specify two optional quality value (QV) constraints. The first constraint QV1 specifies when a mismatch should be ignored. That is, a mismatch is ignored if and only if the sum of the Phred scores of the two mismatching bases is lower than the specified QV1 threshold value. The second constraint QV2 specifies when mismatches should be regarded as critical difference in clustering. That is, two sequences are joined in a cluster only if the sum of the Phred scores of all their mismatching bases is below the $QV2$ threshold value. Therefore, $0 \leq QV1 \leq 93 \times 2$ and $0 \leq QV2 \leq 93 \times 6$ in this paper since our similarity threshold allows at most 3 mismatches (Cock *et al.*, 2010). Note that using SEED with the QV information results in a larger memory footprint, because the Phred scores of all sequences need to be read into memory. Since filtering the sequences by quality prior to the clustering may be often an attractive alternative, QV is an optional parameter

in the SEED program.

2.2.6 SEED System Design

(A) General Features. SEED has been implemented in C++ as a standalone cross-platform tool for Linux, OS X and Windows operating systems. It expects sequences formatted in standard FASTQ format. It can be run in the three modes *ordinary*, *fast* and *short*. The ordinary mode uses block spaced seeds of weight 12 as listed in Table A-2 and supports read sequences of length 36-100 bps. The fast mode uses block spaced seeds of weight 13 and supports sequences of length 58-100 bps. The short mode uses block spaced seeds of weight 6 and supports sequences as short as 21 bps. The fast mode provides the fastest processing time, but requires long sequences and slightly more memory than the ordinary mode. The short mode is suitable for small data sets of short sequences like miRNA sequences, but it is slower than the ordinary mode. The default setting is the ordinary mode.

(B) Performance Optimization. To optimize the time and memory performance of SEED, we have implemented the following features.

Memory Performance

- Each base stored in memory corresponds to two bits.
- Only one copy of each sequence is stored in memory, while the hash tables store pointers to all duplicates.
- The pointers are integer offsets, requiring 4 bytes each instead of 8 bytes on a 64-bit machine.

Time Performance

- A garbage collection is performed in short intervals to prevent long chaining events. Pointers to already processed sequences that have been assigned to clusters are discarded.

- A different set of block spaced seeds of weight 13 is used in the fast mode for sequences of lengths longer than 58 bps. The 1's in the spaced seeds are positioned as close to the $3'$ ends as possible. The latter results in more evenly distributed sequences in the hash table and reduces the bucket sizes. This is important because the read quality near the $3'$ end is usually lower, which could be the cause of mismatches among sequences belonging to the same cluster.

2.3 Evaluation

2.3.1 Test Results with Simulated Data

To test the performance of SEED, we generated 1000 random center sequences. For each of these, we randomly generated sequences with mismatches and overhanging ends, so that the number of center sequences was the number of true clusters. The main objectives of these tests were to determine how well SEED clusters the sequences with respect to the number of clusters, and the number of falsely assigned members in them compared to the true clusters. In the following, the latter aspect is referred to as the false positive ratio (FPR), which is the number false positive members divided by the size of a cluster averaged for all clusters in a set. In addition, the same tests were used to empirically determine the time

and memory performance of the algorithm. In each test we changed only one parameter while keeping the remaining parameters constant. The results of these tests are presented in Tables A-3a to A-3g. They include tests for the number of sequences, the sequence length, the number of true clusters, the number of mismatches, the number of overhanging ends and the QV1/QV2 constraints, respectively. The QV mode of the program was only used for the corresponding tests in Tables A-3f to A-3g.

The time to cluster with SEED 10-100 million sequences of 40 bp in length increases linearly from 24 to 233 minutes, respectively (Table A-3a). For the same data set, the memory footprint increases only sub-linearly from 2.6 to 8.0 GB. When clustering sequence sets of increasing lengths, then the time also increases linearly, while the memory usage shows no change (Table A-3b). With increasing numbers of true clusters the time requirement also changes sub-linearly and the memory usage stays almost constant (Table A-3c). The number of clusters with at least 5,000-10,000 members assembled by SEED is consistently smaller than the number of true clusters in the test data sets (Tables A-3a to A-3g). However, the FPR in the cluster sets is almost exclusively 0. This means that SEED tends to split true clusters into smaller ones, but without contaminating them with false positive members from other clusters. This behavior is extremely important for many practical applications, because false cluster assignments would result in information loss, while splitting the clusters into smaller ones will not remove any important sequences. For instance, in assembly projects removing redundant sequences will help to reduce the memory requirements, but when the clusters are contaminated with false positives then the clustering will remove many sequences that may be important for an optimal assembly. Due

to the more incremental similarity transitions among clusters in real data sets, one would expect there higher FPRs than with simulated data. This can be seen in the subsequent tests on real data sets. However, the FPRs on real data sets are still impressively low (see section 3.2).

More mismatches require extra memory for bucket allocation, but the compute time decreases due to shorter chains (Table A-3d). The number of clusters shows the same trend, because the similarity threshold decreases with the number of mismatches allowing more sequences to be assigned to clusters. For similar reasons, the memory requirements shown in Table A-3e grow with increasing numbers of overhanging residues. However, the time requirements are increasing in this case, because the relative differences among the sequences dominate the clustering time. Also, the number of large clusters decreases, because more shifts tend to reduce the cluster sizes.

When running SEED in the quality aware QV mode (see Tables A-3f and A-3g) then the quality scores need to be imported into the clustering process, which increases its memory footprint by about 15%. The time requirements decrease with increasing threshold values of QV constraints, because greater threshold values tend to assign more sequences to clusters in each pass. In case of QV1, the number of large clusters increases, because more sequences can be assigned to clusters for greater QV1 values.

2.3.2 Test Results with Real Data

(A) Data Sets and Experimental Design. The performance and utility spectrum of SEED for real data was tested on four different types of NGS data that were downloaded form

NCBI's Sequence Read Archive (SRA). In all cases the sequence data were based on Illumina's NGS technology. They included experiments from the following application areas: genome resequencing (sample SRX016064 from *Rhodobacter sphaeroides*), ChIP-Seq (samples SRR038848-SRR038851 from *Arabidopsis thaliana*; Kaufmann *et al.*, 2010), RNA-Seq (samples SRR064165-SRR0641 from *Arabidopsis thaliana*; Jiao and Meyerowitz, 2010), and small RNA-Seq (samples SRR032112-SRR032115 from *Arabidopsis thaliana*; Hsieh *et al.*, 2009). The ChIP-Seq data set was used to compare SEED with other clustering methods. Both the genome resequencing and the RNA-Seq data sets were used to evaluate the utility of SEED for *de novo* genome and transcriptome assembly projects with respect to improvements of the memory footprints and the contig sizes of the final results. Another test included a small RNA data set for evaluating SEED's efficacy in identifying clusters of mature miRNA sequences in the absence of a reference genome.

In most test experiments the NGS data sets were clustered with SEED. Subsequently, the resulting center sequences were used as input data sets for the downstream analysis steps that are commonly used in different application fields, such as assembly and genome/transcriptome alignment steps. The final results were then compared to results obtained without SEED preprocessing.

(B) Cluster Quality Tests. To evaluate how well SEED clusters NGS data, we designed test experiments with real data sets where we benchmarked its performance against the "true" clusters obtained from genome alignment results. For comparison purposes, we also included the clustering software UCLUST and the assembly tool SSAKE in these tests (Warren *et al.*, 2007b; Edgar, 2010). The former was chosen as a software representative

with utilities similar to SEED's. In contrast to this, the typical use case of assembly tools is different, but when they are run on short reads with very stringent overlap criteria then they can fulfill in parts the utility requirements of an NGS clustering tool. Among the many assembly tools available, SSAKE was chosen here because its output format provides the read positions in the contigs which simplified the downstream post-processing of the results. As test data, we used the four ChIP-Seq sets from *Arabidopsis thaliana*. These samples were selected because ChIP-Seq data contain highly variable enrichments of read pileups (peaks) along the chromosomes which is a relatively realistic and also challenging situation when testing the performance of a NGS clustering tool. The true clusters for these data sets were obtained by aligning the reads with Bowtie against the *Arabidopsis* reference genome while allowing up to three mismatches in the alignments. Subsequently, all aligned sequences that completely overlapped with other sequences in the pileup were assigned to clusters with two or more members. Sequences with no or only partial overlaps to other reads were assigned to singlet clusters. The resulting data set is referred to as the "true" cluster set, because it resembles an almost ideal benchmark result of high quality. To obtain meaningful results for the other tools, we used for them comparable parameters. SEED clustering was run with up to three mismatches, but no overlapping ends to match the constraints of the alignment-based reference cluster set. For UCLUST we used comparable parameters by setting the identity parameter to $\frac{l-k}{l}$. Similarly, SSAKE was run with settings that were optimized to obtain only clusters of almost identical sequences. Most importantly, its parameter for the number of matched positions was set to $l - k$.

Table 2.1 gives an overview of the clustering results obtained by the different

Table 2.1: Clustering with different methods. The clustering results for four ChIP-Seq samples (a-d) are shown for the true clusters (alignment based method), SEED, SSAKE, and UCLUST with and without its *optimal* mode. The "true" cluster data were used as references to compute the Jaccard index in the fourth column.

Method	# clusters	# clusters identical with true ones	Jaccard index	Time	Memory in GB
		(a) SRR038848 (4,962,666 reads aligned)			
True	1,106,780				
SEED	973,627	632,209	0.96	00:06:12	2.3
UCLUST	977,904	618,101	0.92	01:28:54	0.4
UCLUSTo	976,871	622,028	0.92	01:44:25	0.4
SSAKE	1,431,122	650,596	0.86	00:20:09	3.0
		(b) SRR038849 (2,435,754 reads aligned)			
True	973,673				
SEED	880,920	512,270	0.97	00:04:02	2.2
UCLUST	873,784	500,982	0.94	00:36:23	0.4
UCLUSTo	873,135	502,654	0.94	00:42:43	0.4
SSAKE	1,070,654	515,574	0.91	00:13:56	2.3
		(c) SRR038850 (5,386,160 reads aligned)			
True	3,365,685				
SEED	3,151,149	664,359	0.95	00:09:47	2.8
UCLUST	3,086,836	669,243	0.88	04:13:09	1.4
UCLUSTo	3,084,657	674,211	0.88	07:12:52	1.4
SSAKE	3,814,607	599,858	0.86	00:51:38	6.9
		(d) SRR038851 (3,148,061 reads aligned)			
True	2,182,354				
SEED	2,038,577	287,903	0.94	00:06:28	2.5
UCLUST	2,096,534	297,756	0.84	01:37:47	0.9
UCLUSTo	2,094,080	300,539	0.85	01:45:00	0.9
SSAKE	2,540,359	214,013	0.77	00:34:10	4.6

methods. Compared to the other methods, SEED has at least a 3- to 10-fold better time performance than the other two methods, but its memory requirements are not as low as UCLUST's. With respect to the cluster qualities, SEED performs consistently better than the other methods by showing the highest Jaccard index values relative to the true clusters. The Jaccard index is a commonly used similarity measure for comparing clustering results, where values close to 0 indicate low similarities and values closer to 1 higher similarities among the evaluated cluster sets. In addition, we used the clustering results presented in Table 2.1 to compare the prediction performance of SEED with the other methods. For

this, we plotted in Figure A-2 the false positive rates (FPR) against the true positive rates (TPR). The FPR is defined as $\frac{FP}{FP+TN}$ and the TPR as $\frac{TP}{TP+FN}$. The individual variables were determined by finding in the results those clusters that show a minimum similarity x to the true clusters. TP is the number of sequences in each cluster contributing to the similarities; FP is the number of sequences in the clusters that do not contribute to the similarities; TN is the number of sequences not in the clusters which should not contribute to the similarities; FN is the number of sequences not in the clusters which should contribute to the similarities. In the resulting graph (Figure A-2), SEED shows the best performance by having consistently the highest TPR values and in most cases lower FPR values as well. The better sensitivity and specificity of SEED is most likely linked to its virtual center sequence for guiding the clustering process. This approach provides relative accurate approximations of the true cluster centers. In this regard UCLUST is less conservative by centering its clusters around a single seed sequence. In addition, SEED is optimized to cluster very similar NGS reads with variable arrangements of mismatch positions. In contrast to this, UCLUST is optimized for detecting a wider range of sequence similarities based on common word matches in its initial search step. This approach is more likely to miss certain high similarity matches that fall below the word size limit of the algorithm. However, the latter feature appears to be less critical, because even when UCLUST is used in its *optimal* mode, where it does not dependent on common word matches (see rows with UCLUSTo in Table 2.1), the performance of SEED is still better.

One concern with the seed algorithm could be that its clustering results may vary depending on which read is chosen first in the random selection process to initialize the

formation of the virtual center sequence of a cluster. To address this, we also performed tests on the four ChIP-Seq data sets from *Arabidopsis thaliana* where we varied the factor f to compute the virtual center sequence as well as the order of reads (data not shown). The quality of the resulting cluster sets was evaluated again with the Jaccard index. With increasing values of f from 1 to 4 the Jaccard index showed only minor differences (< 0.01) for the four data sets. We set $f = 2$ as the default value in all of our experiments, since it gave one of the best results in our tests and it is also a reasonable choice based on the discussion in Section 2.5. Similarly, changing the orders of reads resulted in insignificant changes of the Jaccard index (< 0.01). These tests indicate a relatively stable performance of SEED with respect to these parameter changes.

(C) Assemblies Assisted with SEED. Assemblies rank among the most challenging computational problems in the NGS field (Birney, 2011). Partially, this is because they tend to be an iterative and time consuming improvement process with highly variable outcomes for different data sets (Miller *et al.*, 2010). Moreover, their memory requirements and execution times are often so extensive that larger data sets can only be assembled on high performance compute systems with considerable CPU and memory resources. To improve this, we tested SEED for upstream processing prior to assembly and then analyzed the time and memory requirements of the assembly step, as well as the qualities of the resulting contigs. The assembly components of these tests were performed with Velvet which is one of the most widely used assembly tools for NGS data (Zerbino and Birney, 2008b; Schmidt *et al.*, 2009). To run the assemblies with optimized parameters, the Velvet Optimiser tool was used. The genome assemblies were performed with Velvet only, and the transcriptome

assemblies included both Velvet and its transcriptome-specific Oases component. All software tools were run on a single CPU core (64-bit 2.4GHz Xeon Quad Core Harpertown) to allow fair comparisons of their time and memory usages.

Genome Assembly

Table 2.2 and Figure 2.1 summarize the assembly results for the genome resequencing data set from *Rhodobacter sphaeroides* with Velvet. These tests were performed with and without SEED preprocessing. A random set was included for comparison, where we assembled the same number of sequences as obtained in the preprocessing step with SEED, but by randomly selecting the reads from the raw data set. Compared to the non-preprocessed data set, the assembly time and memory requirements in the SEED data set are greatly reduced by 84.8% and 41.2%, respectively (Table 2.2a). With respect to the quality of the assembly results, several commonly used quality measures improved in the SEED data set compared to the non-preprocessed data set: the number of contigs decreased by 14.0%, the mean length of the contigs increased by 16.8% and the N50 value increased by 26.5%. The latter is the contig length where 50% of the entire assembly is contained in contigs of at least this value. In contrast to this, the corresponding measures in the data set generated by random sampling show the opposite trend. A more detailed overview of the cluster size distributions in the three result sets is given in Figure 2.1. In this plot, the SEED data set shows in comparison to the other tests the highest cumulative contig sizes.

Transcriptome Assembly

To also test whether SEED preprocessing could provide improvements for assemblies of transcriptomes, we performed similar tests with the chosen RNA-Seq data set from *Ara-*

Table 2.2: Assembly tests. The assembly results with Velvet/Oases are shown for (a) the genome resequencing data set from *Rhodobacter sphaeroides* and (b) the transcriptome RNA-Seq data set from *Arabidopsis thaliana*. The table compares row-wise the results for the following preprocessing steps of the raw sequences: no preprocessing, preprocessing with SEED, random sampling of the same number of reads obtained with SEED. The parameters used for SEED were ≤ 3 mismatches, ≤ 3 overhanging ends and QV mode disabled. The corresponding cluster size distributions for the genome assembly in part (a) are given in Figure 2.1.

Preprocessing	# sequences to assemble (Read length)	# contigs	N50	Mean length of contigs	Memory for assembly	Time for assembly	Memory for clustering	Time for clustering
			(a) Genome Assembly					
None	51,448,694 (36bp)	2,230	5,143	2,039	9.7 GB	07:53:54	-	-
SEED	10,644,813 (36bp)	1,918	6,504	2,382	5.7 GB	01:11:59	4.1 GB	03:41:29
Random sampling	10,644,813 (36bp)	2,924	3,855	1,531	2.5 GB	01:12:25	-	-
			(b) Transcriptome Assembly					
None	72,295,211 (37bp)	21,014	452	338	28 GB	15:08:36	-	-
SEED	29,841,222 (37bp)	12,988	507	391	22 GB	05:59:33	8.7 GB	04:09:51
Random sampling	29,841,222 (37bp)	12,868	396	315	12 GB	05:57:09	-	-

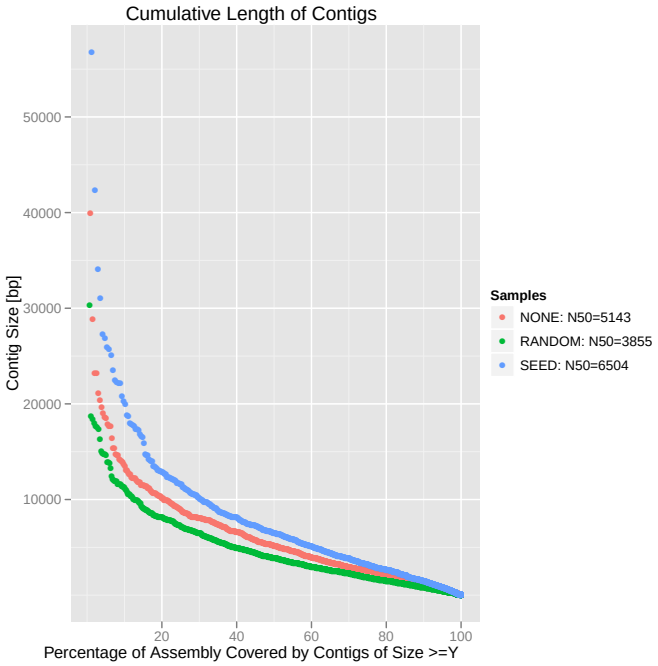


Figure 2.1: Cumulative contig sizes of genome assemblies. The plot compares the cumulative contig size distribution of the Velvet assembly results presented in Table 2.2a (for details see table legend). In this plot, the N50 value is the contig size (Y-axis) at 50% of the assembly coverage (X-axis).

bidopsis thaliana. When using SEED, both the time and memory requirements decreased by 60.4% and 21.4%, respectively. In addition, the mean contig length and the N50 value could be increased by 15.7% and 12.2%, respectively.

The above results on genome and transcriptome data clearly indicate that SEED preprocessing can improve the performance of downstream sequence assemblies using Velvet with respect to compute time, memory usage and quality parameters of the final contigs. Time and memory improvements are the main advantages here, whereas quality enhancements of the final results are likely to vary depending on the specific challenges presented by different sequence types. Investigating which data sets are particularly affected by this and

how SEED exactly improves the quality of assemblies (*e.g.* error correction), goes beyond the scope of this study. When assembling transcriptome data, SEED clustering will help to reduce the extreme redundancies of very abundant mRNA species in these data sets, while maintaining the important information relevant for many RNA-Seq applications. On the other hand, when assembling genomes with highly repetitive sequences, then it will be often necessary to perform SEED preprocessing with very stringent mismatch settings (*e.g.* $k \leq 1$), because higher numbers of mismatches in SEED clustering may eliminate information critical to achieve an optimal assembly of highly similar genomic regions.

(D) Discovery and Profiling of miRNAs with SEED. To explore the potential utility of SEED for identifying and profiling mature miRNA clusters in unsequenced organisms, we performed the following tests. First, we clustered with SEED the raw sequences from four different NGS samples from a recently published small RNA profiling study in *Arabidopsis thaliana* (Hsieh *et al.*, 2009). In this study the authors determined by NGS the expression profiles of 180 miRNAs from root and shoot tissues both grown in the presence and absence of phosphate (Pi). Subsequently, we identified for all miRNAs profiled in the published study the corresponding center sequences in the SEED clustering results. In this association step, the center and mature miRNA sequences had to fully overlap and show not more than one mismatch. Finally, we compared the sequence counts (expression profiles) for each of the miRNAs in the published data set with the size of the corresponding SEED clusters (Table 2.3). Considering only clusters with at least 10 sequences, 76.1-89.4% of the miRNAs in the published data set could be associated with SEED clusters. The likelihood of finding this many overlaps just by chance is very low (random sampling test $p < 10^{-5}$). On

average, these clusters contain 20-48% more sequences than clusters obtained by a simple counting approach of absolutely identical reads (data not shown). The Pearson correlation coefficients (PCC) for the sequence counts for each miRNA in the published data set and the corresponding SEED clusters are for all four samples relatively high (PCC: 0.82-0.91). This high correlation, and the high coverage of known miRNAs detected by these tests, illustrate SEED's utility for identifying in unsequenced genomes candidate clusters of mature miRNA sequences and obtaining for them relatively reliable expression data. A challenge in real data sets without a reference genome will be the identification of the correct miRNA clusters among the much larger pool of unrelated clusters (third column in Table 2.3). This can be largely overcome by sequence similarity searching. Here one can identify clusters with similarities to known miRNAs, which are often evolutionary conserved. In addition, one can easily eliminate by similarity searching against reference databases the typical contaminants in small RNA data sets, such as ribosomal RNAs or transposons.

Table 2.3: miRNA profiling with SEED. The table gives for the four small RNA samples from Hsieh *et al.* (2009) the number of sequences in each data set, the number of clusters obtained by SEED with ≥10 members, the relative number of miRNAs covered by these clusters, and the Pearson correlation coefficients for the published read counts and the ones obtained by SEED.

Samples	# Sequences	# Clusters (size ≥10)	miRNAs identified (all samples 96%)	PCC
SRR032112 (Root -Pi)	5,142,120	37,315	76.1%	0.91
SRR032113 (Root +Pi)	4,919,514	38,193	83.3%	0.89
SRR032114 (Shoot -Pi)	4,862,947	46,776	89.4%	0.82
SRR032115 (Shoot +Pi)	5,003,481	43,176	86.6%	0.87

Chapter 3

BRANCH Algorithm

3.1 Introduction

This study proposes a new method, named BRANCH, for improving the completeness of *de novo* transcriptome assemblies by making use of partial or complete genomic sequence information from the same or closely related species. It involves the initial *de novo* assembly of the RNA-Seq reads to transfrags and DNA reads to genomic contigs using existing NGS assembly software for both types of data. For instance, the genomic reads can be assembled with Velvet (Zerbino and Birney, 2008a), ABySS (Simpson *et al.*, 2009), ALLPATHS (Butler *et al.*, 2008), SOAPdenovo (Li *et al.*, 2010; Luo *et al.*, 2012) or IDBA (Peng *et al.*, 2010), while the RNA reads can be assembled with *de novo* transcriptome assemblers like Velvet/Oases (Zerbino and Birney, 2008a; Schulz *et al.*, 2012), Trinity (Grabherr *et al.*, 2011), Trans-ABySS (Robertson *et al.*, 2010), SOAPdenovo-Trans (Xie *et al.*, 2013) or T-IDBA (Peng *et al.*, 2011). In a downstream transcriptome assembly enhancement step, the

genomic contig information is used to identify novel exons, extend incomplete transfrags and join fragmented ones using the BRANCH algorithm introduced in this study.

BRANCH contains features that intersect in parts with reference-based splice variant assembly tools (sometimes referred to as *ab initio* assemblers; Trapnell *et al.*, 2010; Guttman *et al.*, 2010; Feng *et al.*, 2010; Li *et al.*, 2011), such as the identification of splice variants from RNA sequence alignments against a reference. What makes BRANCH distinct from these tools is that it is designed to maximize the number and completeness of exons contained in preassembled transfrags guided by partial or complete genome sequences from the same or a closely related organism. It does this even for sequence regions with low RNA read coverage. This functionality is novel and relevant for *de novo* transcriptome assembly projects of unsequenced or only partially sequenced genomes, because the additional exonic sequence information will contribute to the functional annotatability of the coding regions of RNA sequences in downstream protein similarity searches.

3.2 Methods

3.2.1 Overview of the Algorithm

BRANCH consists of two major components: *Exon Detection* and *Transfrag Extension*. The *Exon Detection* component aligns the RNA reads against the preassembled *de novo* transfrags, and then it aligns both the transfrags and the remaining reads (that failed to align) against preassembled genomic contigs or a closely related genome using a modified version of the BLAT alignment program (see the discussion below; Kent, 2002). Subse-

quently, it identifies exons and splice junctions in the read pileups against the contigs. Pileup regions meeting certain minimum length and read coverage requirements are considered exons, and low coverage regions between them are introns if they are spanned by gapped alignments and splice junction signals. In addition to the exons contained in the initial transfrags, this step identifies novel candidate exons that are often missed in *de novo* transcriptome assemblies, mainly due to uneven RNA read coverage. Guided by the additional DNA sequence information, BRANCH is designed to resolve those low coverage regions very efficiently. The *Transfrag Extension* component builds a weighted directed acyclic graph (DAG) where the nodes represent the detected exons and the edges splice junctions, while recording the paths through the graph corresponding to each transfrag. The weight of an edge is determined by the read density supporting the connectivity between the nodes. It then extends the recorded paths (*i.e.* transfrags) by finding the minimum number of paths with the minimum total weight that cover all recorded paths as well as the remaining nodes (*i.e.* the novel exons), resulting in extended transfrags.

The following describes the BRANCH algorithm in more details. Section 2.2 introduces the BLAT-based alignment method, and Sections 2.3 and 2.4 describe BRANCH's exon detection and transfrag extension algorithms, respectively. Some illustrations of the algorithms are given in Figures 3.1 and 3.2.

3.2.2 Alignment Steps

An important preprocessor for our method is an alignment tool that can accurately align short RNA reads as well as much longer transfrags against genomic contigs while inserting

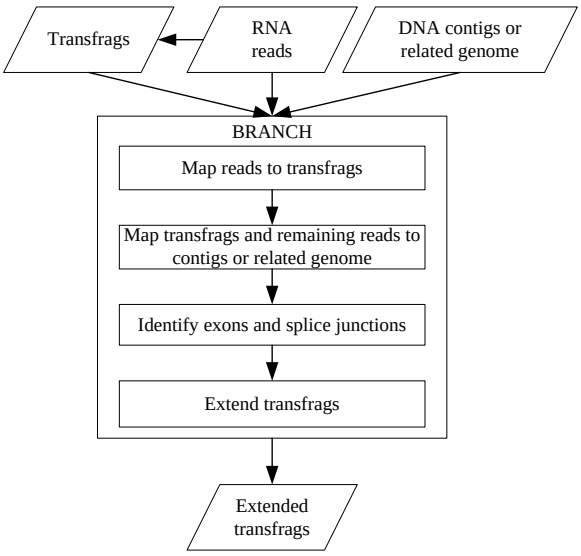


Figure 3.1: Input, processing steps and output of BRANCH. RNA reads are assembled with existing assembly software to *de novo* transfrags. BRANCH maps the RNA reads to the transfrags, and the transfrags and the remaining RNA reads to the genomic sequences. The latter are usually custom assembled contigs or gene sequences from a related organism. Guided by the resulting read pileups, BRANCH identifies existing and novel exons and splice junctions, and uses this information to extend the initial transfrags.

gaps at exon-intron junctions. Several alignment tools are available for mapping short RNA reads with gaps and limited numbers of mismatches against genome sequences. These include TopHat (Trapnell *et al.*, 2009; Langmead *et al.*, 2009b), GMAP (Wu and Watanabe, 2005), SpliceMap (Au *et al.*, 2010) and MapSplice (Wang *et al.*, 2010). For aligning longer transfrag sequences, software tools designed for generating long gapped alignments, such as BLAT, are more suitable than short read aligners. Hence, the current implementation of BRANCH uses a modified version of BLAT that we have optimized to align both types of RNA sequences with acceptable run time, sensitivity, and error tolerance against genomic contigs. These changes to the BLAT executable are similar to those introduced by Grant *et al.* (2011), but they have been customized for our specific needs of aligning long and short

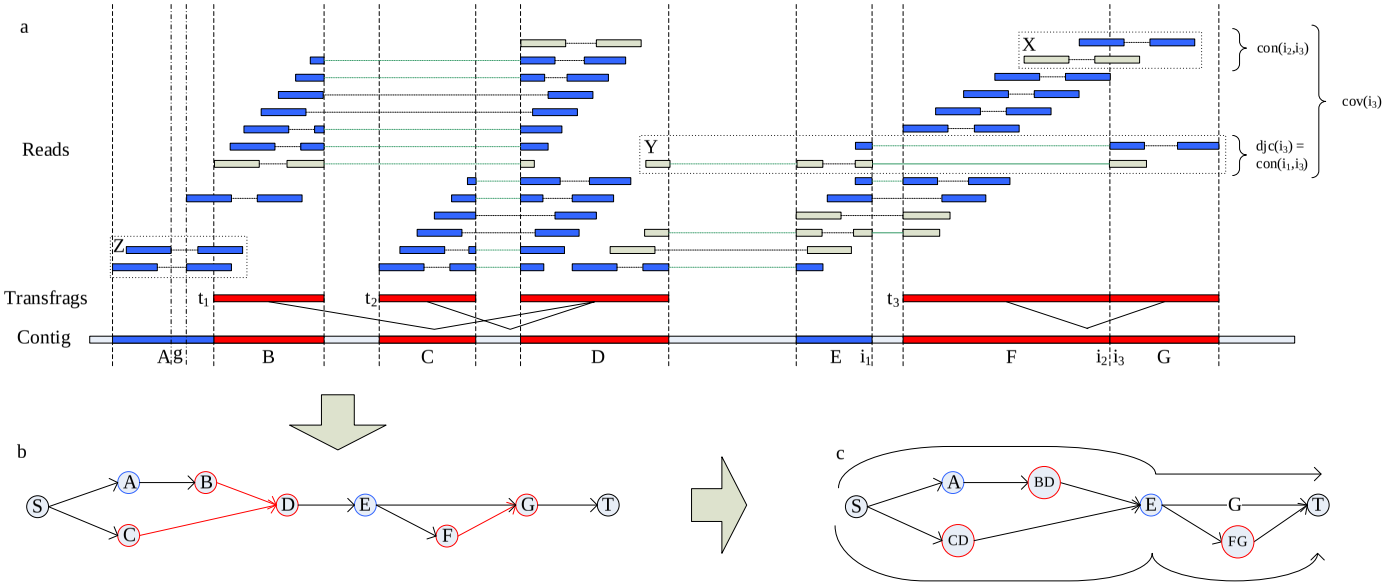

a
Reads
Transfrags
Contig
Z
X
Y
t1
t2
t3
A
g
B
C
D
E
i1
F
i2
i3
G
con(i2,i3)
cov(i3)
djc(i3) =
con(i1,i3)
b
S
A
B
C
D
E
F
G
T
c
S
A
BD
CD
E
G
FG
T

Figure 3.2: (Continued from previous page) Illustration of important features of BRANCH algorithm. (a) A sample pileup is shown containing paired-end (PE) RNA reads, preassembled transfrags (t_1 to t_3) and one contig with exons A to G. Reads of good quality are indicated in blue and low quality reads in grey. PE pairs and spliced read fragments are connected with thin black and green lines, respectively. The red bars (B, C, D, F and G) in the contig are the exons identified by BRANCH based on the alignment of the preassembled transfrags against the contig. The blue bars (A and E) in the contig are two additional exons identified by BRANCH based on spliced and PE reads aligning with both contig regions covered by transfrags and those not covered by transfrags. Those exons (here A and E) are often missed by *de novo* transcriptome assemblers due to insufficient read coverage and/or sequence errors. The sequencing gap g in exon A could be closed with PE reads in rectangle Z, because their insert size obtained from the alignment against the contig, agrees with the expected insert size of the library. Another situation where BRANCH improves transcriptome assemblies is given on the right side of the diagram. Here the exon region FG, corresponding to transfrag t_3, is subdivided by an internal splice site i_1/i_2 into two exons. This is supported by a minimum number of splice junction reads with gaps (rectangle Y) spanning contig positions i_1 and i_3. The coverage $\mathrm{cov}(i_3)$ is the number of junction reads overlapping with base position i_3, here reads in rectangles X and Y; the downstream junction coverage $\mathrm{djc}(i_3)$ is the number of junction reads overlapping with base positions i_1 and i_3 in rectangle Y where $i_1+1 < i_3$; and the connectivity $\mathrm{con}(i_2, i_3)$ between positions i_2 and i_3 is the number of reads overlapping with bases i_2 and i_3 in rectangle X where $i_2 < i_3$. (b) A junction graph has been constructed from the alignment. In this graph, exons are nodes and edges are connections among them that are weighted based on the read support from the spliced alignments. Source and sink nodes are added at the beginning (S) and the end (T) of the graph, respectively. The paths corresponding to the *de novo* transfrags are marked in red: $B \to D$, $C \to D$ and $F \to G$. (c) The *Transfrag Extension* Algorithm collapses these paths to path nodes BD, CD and FG. The resulting Minimum weight Minimum Path Cover with given Paths (MMPCP) in the original graph (b) includes the paths indicated by round arrows: $S \to A \to BD \to E \to G \to T$ and $S \to CD \to E \to FG \to T$. Each of them corresponds to an extended transfrag.

sequences. They include early filtration of candidate alignments to minimize execution time, disk space, and support for handling paired-end read data. In addition, the boundaries of identified introns are screened for the presence of canonical (GT-AG) and non-canonical (*e.g.* GC-AG, AT-AC) splice sites. This information is used to optimize the exon-intron junctions obtained from the alignment results.

3.2.3 Exon Detection Algorithm

The *Exon Detection* (ED) Algorithm identifies exons and splice junctions. It uses the modified BLAT software described in the previous section to first align the RNA reads (single or paired-end) against the transfrags, and then the transfrags as well as all the remaining reads (not mapped in previous step) against the contigs or a related genome reference. The latter read pool contains RNA reads derived from exon sequences missing in the transfrag sequences, while others may have failed to align due to base calling errors. After aligning the transfrags and reads to the contigs, the ED Algorithm identifies exons and splice junctions guided by the coverage information obtained from the alignment result. Regions with a minimum RNA read coverage b and a minimum width a are considered exons. Both a and b are user definable parameters. In future upgrades of BRANCH these thresholds will be optimized for the user dynamically to minimize false positive exon predictions due to contaminations with unspliced pre-mRNAs and other sources of noise in the data. After identifying candidate exons, the algorithm locates splice junctions between them based on the gap positions in the tranfrag sequences and/or RNA junction reads aligned against the contig sequences. Alternative splice sites within exons are identified in areas where

a minimum number of junction reads share the same gap that spans across one or more exon regions. Figure 3.2a illustrates these steps with an example. The outcome of the ED Algorithm are additional exonic sequences not contained in the initial transfrag sequences. This includes extensions of incomplete exons and the identification of novel exons (complete or partial) along with their connections. The detailed steps of the ED Algorithm and its pseudo code are given below.

Step 1 is the alignment of the RNA reads, transfrags and contigs as described above.

Step 2 identifies an exon region based on the alignment, where we denote the coverage of a contig position i by $\mathrm{cov}(i)$. In Figure 3.2a, the coverage of junction base i_3 is the number of junction reads overlapping with it in rectangles X and Y. The reads in rectangle X align over their full length against the transfrag t_3 and the contig, whereas the reads in rectangle Y align completely only against the contig. Both read sets overlap with position i_3. The algorithm computes the coverage for each contig base, and identifies any contig region, with start and end positions $[l, r]$, as an exon range, if the width of the contig region satisfies $r-l+1 \geq a$ and the average coverage of the contig region satisfies $\sum_{l \leq i \leq r} \frac{\mathrm{cov}(i)}{(r-l+1)} \geq b$, where a and b are the minimum width and the minimum coverage requirements, respectively. In certain cases the newly identified exon regions may be fragmented in areas with very low or no RNA read coverage. Suppose a novel exon range $[l, r]$ contains sufficient read coverage in subranges $[l, i]$ and $[j, r]$ $(i < j)$, but subrange (i, j) has zero coverage. In such a case two partial exons $[l, i]$ and $[j, r]$ will be identified instead of the complete exon $[l, r]$. Such gaps can be closed, if there is a sufficient number of PE reads spanning $[l, i]$ and $[j, r]$, and the mapping distances of the read pairs agree with the approximate insert length of the library.

An example of such a case is given in Figure 3.2a, where the coverage gap g divides exon A into two parts, but it can be closed with the PE read support shown in rectangle Z. To minimize the risk of incorporating introns, this type of gap closures are only performed if the mapping distances of the read pairs agree with the approximate insert length of the RNA-Seq library. Alternatively, the user can specify this parameter.

Step 3 identifies alternative splice junction sites within exons. Here, we define the *upstream* and the *downstream junction coverage*. The upstream junction coverage at contig position i, denoted as ujc(i), is the number of reads having bases at positions j and $j+1$ aligned at contig positions i and $k > i+1$, respectively. Similarly, the downstream junction coverage at contig position i, denoted as djc(i), is the number of reads having bases at positions $j-1$ and j aligned at contig positions $k < i-1$ and i, respectively. For example, the downstream junction coverage at base i_3, djc(i_3), is the number of junction reads in rectangle Y of Figure 3.2a covering i_3. The aligned junction reads overlap with bases i_1 and i_3, where $i_1+1 < i_3$. The algorithm records the upstream and downstream junction coverages for each contig base, and then splits such a region $[l, r]$ at contig positions i and $i+1 (l \leq i < i+1 \leq r)$, if the upstream junction coverage at i satisfies ujc(i) $\geq c$ or the downstream junction coverage at $i+1$ satisfies djc($i+1$) $\geq c$, where c is the minimum upstream/downstream junction coverage requirement to split exon regions.

Step 4 determines which exons are joined based on their *connectivity* in the alignment result. The connectivity between the last base of an exon and the first base of a downstream exon at contig positions i and $j > i$, denoted as con(i, j), is the number of reads having bases at positions k and $k+1$ aligned at contig positions i and j. In Figure 3.2a, the connectivity

between i_2 and i_3 is the number of reads in rectangle X with matching bases at positions i_2 and i_3. The algorithm computes the connectivity for each pair of exons and identifies two exons at positions $[l_1, r_1]$ and $[l_2, r_2](r_1 < l_2)$ as a junction, if the connectivity of the pair of boundary bases at r_1 and l_2 satisfies $\text{con}(r_1, l_2) \geq d$, where d is the minimum connectivity requirement to connect two exons.

3.2.4 Transfrag Extension Algorithm

The *Transfrag Extension* (TE) Algorithm extends and often joins *de novo* transfrags based on the additional exon sequences and splice junctions identified in the previous *Exon Detection* step. For this it identifies the connections best supported by the data and then joins the corresponding sequence fragments accordingly. The final output is extended transfrag sequences, as well as novel transfrags. For example, if the connectivity data obtained in the previous step indicate that a newly identified exon ϵ is connected with an existing exon ϵ', and ϵ' appears in two separate transfrags t and t', then the algorithm has to decide if ϵ is connected with t and/or t'. A similar but not identical problem is solved by the Cufflinks algorithm for identifying transcript variants in RNA-Seq data (Trapnell *et al.*, 2010). Thus, our algorithm adopts certain components of this method, while others are specific to BRANCH's main application addressing the transfrag extension problem.

(A) Mathematical Formulations

Definition 2. *A junction graph is a DAG, where each node represents an exon and each edge represents a splice junction.*

Based on the exons and splice junctions identified by the ED Algorithm, BRANCH

Algorithm 2 Exon Detection: ED(R, T, C)

1: Align reads R to *de novo* transfrags T with BLAT, and then align T and the unaligned reads R_n to contigs C
2: Record the coverage for each base at contig position i, and identify each region $[l, r]$ in a contig, where $r-l+1 \geq a$ and $\sum_{l \leq i \leq r} \frac{cov(i)}{(r-l+1)} \geq b$
3: Record the upstream and downstream junction coverages for each base at contig position i, split a region $[l, r]$ at bases i and $i+1 (l \leq i < i+1 \leq r)$, if ujc$(i) \geq c$ or djc$(i+1) \geq c$, and identify the resulting regions as exons
4: Record the connectivity for each pair of bases at contig positions i and $j > i$, and identify the splice junction of each exon pair $[l_1, r_1]$ and $[l_2, r_2] (r_1 < l_2)$, if con$(r_1, l_2) \geq d$

builds a *junction graph* G where each node v represents an exon ϵ and the connecting edges are splice junctions among exons. Two nodes v and v' are connected by an edge $e(v, v')$ if their corresponding exons ϵ and ϵ' are junction exons. Similar to the approach chosen by Trapnell *et al.* (2010), the graph is weighted based on the *percent-spliced-in* value introduced by Wang *et al.* (2008). The latter expresses the density of the RNA reads supporting a transcript relative to the density of all the RNA reads mapping to the corresponding genomic region of the transcript. The percent-spliced-in value for any exon ϵ (and thus node v in the junction graph) is defined by:

$$\psi_\epsilon = \frac{\text{number of compatible reads overlapping with exon } \epsilon}{\text{number of reads overlappling with exon } \epsilon \times \text{length of exon } \epsilon}. \quad (1)$$

In the above formula, the *overlap* and *compatibility* of an aligned RNA read γ and an exon ϵ are defined as follows. Read γ and exon ϵ overlap if and only if their start coordinates $l(\gamma)$ and $l(\epsilon)$ and end coordinates $r(\gamma)$ and $r(\epsilon)$ in the reference genome satisfy $l(\gamma) \leq l(\epsilon)$ and $r(\gamma) \geq l(\epsilon)$, or $l(\epsilon) \leq l(\gamma)$ and $r(\epsilon) \geq l(\gamma)$. Overlapped read γ and exon ϵ are compatible if and only if any gap $[i(\gamma), j(\gamma)]$ in the alignment of γ does not overlap with the exon ϵ. The value for any exon pair ϵ and ϵ' (and thus edge $e(v, v')$) is defined as the absolute difference of their weights with amplification:

$$w(\epsilon, \epsilon') = -log(1 - |\psi_\epsilon - \psi'_\epsilon|) \quad (2)$$

The smaller w is, the more likely that the ϵ and ϵ' are from the same transcript.

Clearly, each given transfrag corresponds to a path in G. These are called *given paths*. Since we are interested in extending the transfrags by possibly merging them and adding more novel exons, we formulate the transfrag extension problem in BRANCH as a combinatorial optimization problem called the Minimum weight Minimum Path Cover with given Paths (MMPCP) problem. An MMPCP is a smallest set of paths with the minimum weight in the junction graph G that contains all the given paths P as subpaths and cover all the nodes of V. Here, we seek the smallest number of paths because we would like to maximize the length of each extended transfrag. The minimum total weight requirement guarantees that any two exons ϵ and ϵ' in each extended transfrag are from the same true transcript.

(B) Outline of the TE Algorithm. Our idea to find an MMPCP is to build a new junction graph G' from G by (1) converting each given path $p \in P$ to a node $v(p)$ and (2) maintaining the connection between any two nodes v and $v' \notin p$ through a subpath of p by introducing an edge $e(v, v')$. The new node $v(p)$ will be referred to as *path node* and the new edge $e(v, v')$ as *path edge*. To keep the two graphs equivalent, the total weight of a given path will be added to each in-edge of the corresponding path node, and the path edges will be weighted using the total weights of the corresponding subpaths. This conversion is illustrated in Figure 3.2b and 3.2c. Then we invoke a *Combinatorial Optimization* (CO) Algorithm for solving the Minimum weight Minimum Path Cover (MMPC) problem in the new graph G' (see Appendix). If P' is the resulting MMPC for G' from the CO Algorithm, the paths in P', or the transfrags they represent, may not be fully extended. To address this, we can

iterate the above process for solving the MMPCP problem by recording P' as new given paths and extending them recursively, until they cannot be extended anymore. The TE Algorithm is more formally outlined in the following pseudocode. The final output of the TE Algorithm consists of transfrags that have been extended with exonic sequences from the Exon Detection step, as well as some novel transfrags.

3.2.5 Implementation and Performance

BRANCH has been implemented in C++ with the LEMON library (Dezso *et al.*, 2011) for Linux operating systems. The modified BLAT executable is distributed along with BRANCH. The expected input includes RNA reads (single or paired-end), assembled transfrags, and genomic contigs or gene sequences from a closely related species. Most of BRANCH's execution time is spent on the initial alignment with BLAT ranging from 0.1-0.5 hours per million reads. The subsequent steps are more memory than CPU intensive for storing the genomic contigs (0.1 GB RAM per million nucleotides). Both the execution time and memory usage of BRANCH are approximately linear in the number of RNA-Seq reads and size of the genomic contigs, respectively.

3.3 Evaluation

3.3.1 Test Results with Simulated Data

(A) Background. The performance of BRANCH was tested with real and simulated data. The main objective of these experiments was to assess the efficiency of BRANCH for improving the representation of full-length transcripts in *de novo* transcriptome assemblies,

Algorithm 3 Transfrag Extension: TE(G, P)

```
Assign weights to the edges of G using Equation (2)
for each given path p ∈ P do
    Convert p to path node v(p) and add the total weight of p to each in-edge of v(p)
    for any pair of nodes v and v' ∉ p do
        if there is a path p' ∉ P from v to q ∈ subpath(p) and then to v' then
            Introduce a path edge e(v, v')
            Weight e(v, v')
        end if
    end for
    Delete p from G
end for
{G is converted to G'}
P' ← CO(G')
if P' = P then
    return the resultant MMPCP P'
else
    return TE(G', P')
end if
```

but also its splice variant resolution, error tolerance, and robustness with respect to variable degrees of incomplete representation of transcript and genomic sequences. While tests on real data provide more reliable results for the performance of an algorithm, simulated data were included here because they allow a more systematic evaluation of a wide variety of data properties than this would be possible with real data only. To mimic in these tests real data as much as possible and minimize bias toward any method, all sequences were randomly sampled from a real genome, meaning they were only partially synthetic. The results on real data sets are given in the next section. In the tests with simulated data, we varied the number of RNA reads, the average length of the contigs, the relative genome coverage by the contigs, and the base call error rates in both the RNA reads and the contigs. Benchmarking BRANCH's main utility - the enhancement of transcriptome assemblies guided by genomic sequences - against other tools is currently not easily possible due to the lack of software designed for this purpose. However, a very informative performance measure is to determine how well BRANCH can improve *de novo* assembled transfrags with respect to their full-

length and gene coverage in a genome. For this we compared the final results generated by BRANCH with the initial *de novo* transfrags that we generated in the tests on simulated data with the Velvet/Oases and Trinity transcriptome assemblers. Velvet/Oases and Trinity were chosen here among other software options (*e.g.* Trans-ABySS, SOAPdenovo-Trans), because of their good sensitivity and precision performance (Zhao *et al.*, 2011).

(B) Data Sets and Tests. The simulated test data sets were randomly sampled from the genome and transcriptome sequences of *C. elegans* provided by Ensembl's FTP site. From the genome sequence we sampled three types of contig sets and from the transcriptome two types of RNA-Seq sets as follows: (1) contigs of variable length of 1, 10, 50 and 100 kbp; (2) contigs with variable coverage of the *C. elegans* genome of 40, 60, 80 and 100%; (3) contigs with variable sequence error rates of 0, 1, 2 and 3% by substituting bases at random positions; (4) different numbers (10, 30, 50 and 70 million) of paired-end RNA reads of 2x 100 bp length and 200-300 bp insert length while maintaining an abundance distribution among the reference transcripts that is typical for RNA-Seq samples (see Table B-1); and (5) RNA reads with variable error rates of 0, 1, 2 and 3%. The simulated RNA-Seq sets were assembled to transfrags using Velvet/Oases with its parameter optimization script and Trinity with its default parameter settings.

To be consistent with recent studies on *de novo* RNA assemblies, we define in our tests *sensitivity* and *precision* in a similar manner. *Sensitivity* is the number of reference transcripts which could be aligned, here with BLAT, to a transfrag with $\geq$95% identity over $\geq$80% of the transcript's length and $\geq$95% of the transfrag's length (Martin and Wang, 2011). Additionally, test results with variable length coverage values are given in section

3.2.3 and Figure 3.4. *Precision* is defined as the percentage of transfrags which could be aligned to a reference transcript with ≥95% identity over ≥95% of the transfrag's length, but without a minimum length coverage requirement for the transcript (Zhao *et al.*, 2011; Schulz *et al.*, 2012; Robertson *et al.*, 2010). Moreover, we compare among the different assembly methods the following performance parameters: numbers of covered transcripts, complete transcripts and completely represented exonic regions of genes. For the latter two we also require ≥95% identity and ≥95% length coverage of the reference and the transfrag.

(C) Results. Figure 3.3 and Tables B-2 to B-6 give the test results for the simulated data sets for variable contig lengths, contig sequence error rates, contig coverages, numbers of RNA reads, and RNA read base call error rates, respectively. All other parameters are constant settings, which are specified in the legends. Compared to the input transfrags generated by Velvet/Oases and Trinity, BRANCH post-processing improves their sensitivity and precision substantially by 2.3-19.9% and 1.7-15.7%, respectively. The relative sensitivity improvements by BRANCH for both assemblers are about two fold higher when the coverage of the genome by contigs is raised from 40% to 100% (Figure 3.3c and Table B-4), whereas increasing sequence error rates from 0-3% in the contigs have a less pronounced impact by reducing the relative sensitivity improvements in the most extreme cases by 20-34% (Figure 3.3b and Table B-3). Because BRANCH also identifies novel transfrags, the initial number of transfrags increases as expected (in these tests by 6.7-72.6%). The 487-5,394 transfrag extension events recorded in the BRANCH results lead to 0.2-9.0% more completely assembled transcripts increasing the number of completely assembled exonic regions of genes by 0.1-7.6%. The latter improvements are less pronounced due to the more strin-

gent full-length criteria applied in these cases. Most importantly, the transfrags processed by BRANCH have a 6.0-18.5% higher coverage of the total number of exons annotated in the *C. elegans* genome than the initial transfrag sets. These results indicate that BRANCH improves the chosen quality parameters of transcriptome assemblies relatively effectively over the range of test variables evaluated in these experiments.

3.3.2 Test Results with Real Data

(A) Experimental Design. The performance of BRANCH on real data was tested with published Illumina NGS samples available in NCBI's Sequence Read Archive (SRA). To generate meaningful test results, it was important to choose here NGS data meeting today's standards for efficient RNA-Seq transcriptome assemblies with respect to read length (>50bp) and paired-end read information. BRANCH's performance on the two main types of genomic guide sequences was evaluated by including in one set of tests custom genomic contigs assembled from NGS reads of the same organism as the RNA reads, and in another case existing genome sequence from a closely related organism (Table 3.1). The influence of the completeness of the genomic sequence information on the performance of BRANCH, was tested by comparing the results guided by assembled contigs with those from complete gene sequences.

Two data sets were chosen from diverse multicellular eukaryotic organisms (*C. elegans* and mouse) to account for splice variants and variable degrees of sequence complexity, and a third one was from a unicellular eukaryotic organism (*S. cerevisiae*) with a densely organized genome and rare alternative splicing. To evaluate the impact of directional in-

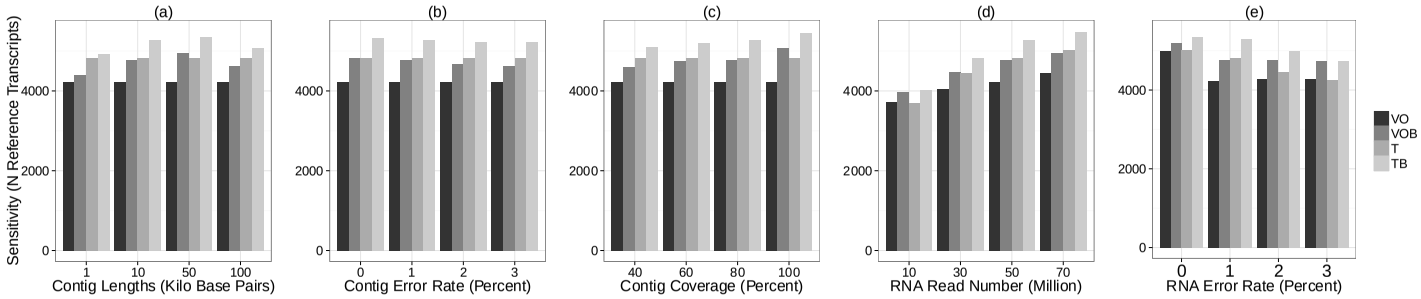


Figure 3.3: Sensitivity tests on simulated data. Sensitivity measures of Velvet/Oases (VO), Velvet/Oases with BRANCH post-processing (VOB), Trinity (T) and Trinity with BRANCH post-processing (TB) are plotted for (a) variable contig lengths, (b) sequence error rates in contigs, (c) relative coverages of the reference genome by contigs, (d) number of RNA reads, and (e) base call error rates in RNA reads. The invariable parameter settings include 10 kbp contig length, 1% sequence errors in contigs, 80% contig coverage, 50 million paired-end RNA reads, and 1% base call error rate in RNA reads.

formation in the RNA reads, we used in two cases non-strand-specific RNA-Seq samples and in another case a strand-specific sample. The RNA reads from all sample sets were assembled with Velvet/Oases and Trinity (Grabherr *et al.*, 2011; Zhao *et al.*, 2011). In case of Trinity the default parameter settings recommended by its developers were used. DNA reads were assembled to contigs with Velvet using the VelvetOptimiser tool for parameter optimization. To also compare against an alignment-based splice variant assembler, we included Cufflinks, which is a fundamentally different method compared to the above *de novo* assemblers. Cufflinks was only included in the test case with the known genome sequences as guide reference, because it was the only situation where the minimal input data types, required for this method, were available. The splice junction information was obtained by aligning the RNA reads with Tophat (Version 2) against the genomic sequences. Both Tophat and Cufflinks were run with their default parameter settings.

The results obtained from the different tests were used to compute similar quality parameters (Table 3.1) as in the previous section assessing among other properties the full-length and splice variant resolution of the transfrags. To also evaluate the functional annotatability of the assembled transcripts before and after processing them with BRANCH, they were used as queries in BLASTX searches (E-value cutoff 10^{-9}) against the protein databases of the corresponding organisms. The obtained results were queried for nearly complete protein matches requiring $\geq$95% identity on the protein sequence level.

(B) Datasets. The first NGS sample set is from *C. elegans*. Its genomic read set contained 57 million 2x 55-76 bp long paired-end reads (accessions: SRR066623, SRR066625; Weber *et al.* 2010) and its RNA-Seq set contained 72 million 2x 100 bp paired-end reads (accession:

SRR316929; Hillier *et al.* 2009). The second sample set is from mouse (*Mus musculus*) with 34 million 2x 76 bp paired-end RNA-Seq reads (accessions: SRR290901, SRR290902; unpublished). The gene sequences from rat (*Rattus norvegicus*) were used in this case as genomic guide sequence to test BRANCH's performance for a situation where a related genome sequence is available. The third sample set is from *S. cerevisiae* with 4 million 2x 76 bp long paired-end genomic reads (accessions: SRR527545, SRR527546; unpublished), and 10 million 2x 76 bp strand-specific paired-end RNA-Seq reads (accession: SRR059177; Levin *et al.* 2010).

(C) Assemblies Assisted with Custom Genome Contigs. The performance test results for the *C. elegans* data are given in Table 3.1a. In comparison to the initial transfrags assembled by Velvet/Oases, BRANCH shows a 6.3% and 10.5% improved sensitivity and precision performance, respectively, when guided by the 88,175 genome contigs assembled for this experiment representing 90.4% of its genome. To evaluate the sensitivity performance over a wider threshold range of transcript length coverage values, Figure 3.4 compares among the different assembly methods the number of reference transcripts from *C. elegans* that aligned with the transfrags over increasing minimum overlap values from $10 - 90\%$. BRANCH exhibits here a consistent improvement compared to the other methods over the full range of overlap thresholds. When comparing the sensitivity performance among the different methods for variable expression levels (see Figure 3.5) then BRANCH shows the greatest improvements for weaker expressed transcripts. This is in agreement with its design feature for improving the assembly of transfrags with low read coverages.

With respect to the other performance parameters recorded in Table 3.1, BRANCH

Table 3.1: Performance on real data. Assembly results of RNA-Seq data from (a) *C. elegans*, (b) *S. cerevisiae* and (c) *M. musculus* are given for the transcriptome *de novo* assemblers Velvet/Oases and Trinity. The splice variant assembler Cufflinks was included in one case where its required input was available. The resulting transfrags were post-processed with BRANCH (*e.g.* referred to as Trinty+BRANCH) using under (a) custom assembled genome contigs[1] or known gene sequences[2] from *C. elegans*, and under (c) the gene sequences from the rat genome[3]. The latter evaluates BRANCH's performance for a case where a closely related guide genome sequence is available. The sample from *S. cerevisiae* (b) uses custom assembled contigs along with strand-specific RNA-Seq data from the same organism. The other two cases contained non-strand specific RNA samples. The acronyms introduced in the first column serve as sample labels in Figures 3.3-3.5. The performance criteria considered in the remaining columns are described in sections 3.1.2 and 3.2.1.

Method	Sensitivity	Precision	N Transf.	N Comp. transcripts	N Comp. genes	N Cov. transcripts	N Exons	N Exten.	N Prot.
(a) Transcriptome Assembly of C. elegans (BRANCH Guided by Genomic Sequences from C. elegans)									
Velvet/Oases (VO)	5,015	32.8%	55,083	3,248	2,986	3,844	96,078	-	3,839
Velvet/Oases+BRANCH (VOB)[1]	5,332	43.3%	62,201	3,446	3,159	4,187	107,467	5,726	4,683
Velvet/Oases+BRANCHg (VOBg)[2]	6,602	42.3%	56,369	4,419	3,973	4,825	107,876	7,696	4,811
Trinity (T)	5,048	39.4%	32,083	3,708	3,416	4,152	116,128	-	4,866
Trinity+BRANCH (TB)[1]	5,303	43.3%	51,997	3,848	3,539	4,360	121,484	5,320	5,706
Trinity+BRANCHg (TBg)[2]	6,309	42.3%	49,197	4,500	4,080	4,852	122,345	6,877	5,892
Cufflinksg (Cg) [2]	5,147	47.7%	14,685	3,300	3,073	3,436	114,029	-	2,997
(b) Strand-Specific Transcriptome Assembly of S. cerevisiae (BRANCH Guided by Genomic Sequences from S. cerevisiae)									
Velvet/Oases (VO)	282	38.6%	75,053	54	54	132	1,211	-	926
Velvet/Oases+BRANCH (VOB)[4]	442	39.8%	80,831	94	94	212	1,875	9,514	1,239
Trinity (T)	315	41.0%	11,451	146	146	201	4,375	-	1,957
Trinity+BRANCH (TB)[4]	412	41.3%	13,394	206	206	261	4,498	2,322	2,119
(c) Transcriptome Assembly of Mouse (BRANCH Guided by Genomic Sequences from Rat)									
Velvet/Oases (VO)	7,103	23.4%	447,689	2,922	2,331	5,230	123,070	-	12,260
Velvet/Oases+BRANCH (VOB)[3]	7,417	24.5%	518,360	3,073	2,478	5,595	123,939	3,325	12,747
Trinity (T)	4,593	25.4%	143,757	1,971	1,621	3,177	100,453	-	8,676
Trinity+BRANCH (TB)[3]	4,916	27.2%	187,478	2,128	1,776	3,501	101,964	1,295	9,217

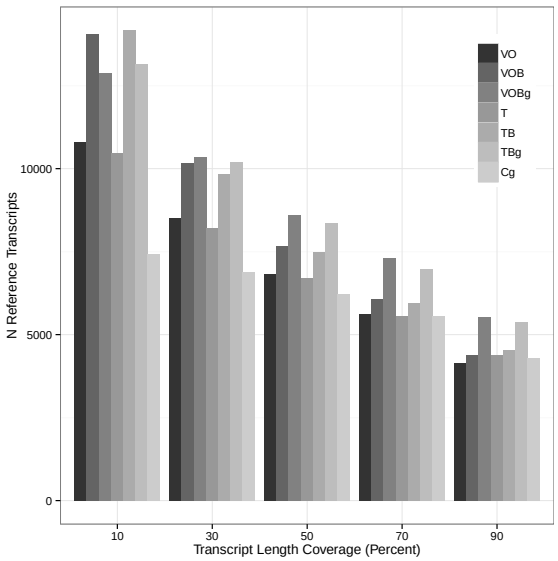


Figure 3.4: Transcript length coverage. The number of reference transcripts of the *C. elegans* data set is plotted that aligned with the transfrags over increasing overlap thresholds from $\geq 10\%$ to $\geq 90\%$. The acronyms assigned to the different methods in the legend are defined in the first column of Table 3.1.

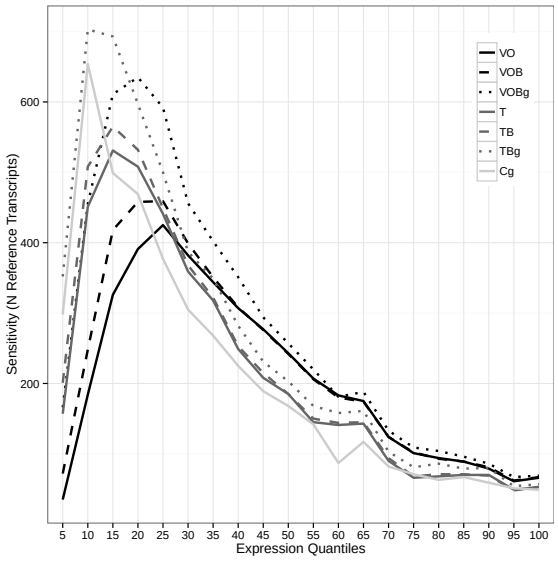


Figure 3.5: Sensitivity performance for variable expression quantiles of *C. elegans* data. The number of assembled transcripts (x-axis) are plotted across different expression levels (y-axis). The acronyms assigned to the different methods in the legend are defined in the first column of Table 3.1.

also increases the number of complete transcripts, complete genes, covered transcripts and exons annotated in the *C. elegans* genome by 6.1%, 5.8%, 8.9% and 11.9%, respectively. When using the transfrags in translated BLASTX searches against the *C. elegans* protein database, the BRANCH results show a remarkable increase (22.0%) of the number of complete protein sequences encoded in the assembled transcript set. In Table 3.1, the number of nearly complete protein sequences is usually larger than the number of complete transcripts, because the latter also contain untranslated 5' and 3' regions that make the full-length cut-off criteria ($\geq$95% sequence identity over $\geq$95% of the reference length) more stringent for transcripts than for proteins. BRANCH processing extends 5,726 transfrags from the initial assembly and it identifies many novel transfrags resulting in a 12.9% increase of the total transfrag pool. When the same tests are performed using Trinity instead of Velvet/Oases as RNA-Seq *de novo* assembler, then BRANCH shows a 5.1% and 3.9% improved sensitivity and precision, and it increases the number of complete transcripts, complete genes, covered transcripts, exons and nearly complete proteins by 3.8%, 3.6%, 5.0%, 4.6% and 17.3%, respectively. Overall these improvements are the result of 5,320 transfrag extension events generated by BRANCH. As expected, when the gene sequences instead of assembled contigs are provided, the transfrags processed by BRANCH show additional improvements compared to the ones obtained from Velvet/Oases. With a total of 7,696 transfrag extension events obtained by BRANCH, the sensitivity improves by 31.6%, the precision by 9.5%, and the number of complete and covered transcripts by 36.1% and 25.5%, respectively. In addition, the number of complete proteins increases by 25.3%. For Trinity the same tests result in similar improvements by BRANCH: the sensitivity, precision, number of complete

transcripts, number of covered transcripts, and the number of complete proteins improve by 25.0%, 2.9%, 21.4%, 16.9%, 21.1%, respectively.

In the test case with known reference genes, BRANCH also outperforms Cufflinks in sensitivity by 22.6-28.3%, but shows a slightly lower precision (5.4%). For the remaining test parameters BRANCH's performance is consistently superior over Cufflinks'. In addition, Cufflinks is unable to produce even nearly as good results (data not shown) in the tests with the other types of genomic guide sequences, BRANCH has been specifically designed for, including custom contigs and genomic sequences from related genomes. The main reasons for these performance difference are as follows. First, Cufflinks has been designed for a different use case, which is the prediction of splice variants for completed genomes, preferentially with well defined gene/exon boundary annotations. Second, due to its transfrag input, BRANCH's performance greatly depends on the quality of the upstream *de novo* assemblies. If those were of poor quality then Cufflinks' ranking in this comparison could change. Third, BRANCH has been optimized to extend transfrags in sequences regions with very low RNA read coverage (Figure 3.5). Despite those utility differences, the performance results presented here demonstrate that it is currently not possible to replace BRANCH's main functionality with a reference-based splice variant assembly tool, even when an "idealized" reference gene set is available as in the Cufflinks result given in Table 3.1a.

(D) Assemblies with Strand-Specific RNA-Seq Data and Custom Genome Contigs. Table 3.1b gives the results for *S. cerevisiae* where strand-specific RNA-Seq data was used along with 1,360 custom assembled guide contigs representing 92.4% of this genome. With this

data set, BRANCH improves the sensitivity and precision of the transfrags generated by Velvet/Oases by 56.7% and 1.2%, and those from Trinity by 30.8% and 0.3%, respectively. At the same time the numbers of complete transcripts, complete genes, covered transcripts and proteins annotated in the *S. cerevisiae* genome increase with BRANCH relative to the input data from both *de novo* assemblers by 41.1-74.1%, 41.1-74.1%, 29.9-60.6% and 8.3-33.8%, respectively. In general, the improvements achieved by BRANCH are more pronounced for the Velvet/Oases input, because Trinity performs better on this data set, leaving less room for improvements. Nonetheless, the results for both *de novo* assemblers demonstrate that BRANCH post-processing can lead to considerable improvements of transfrags generated from strand-specific RNA-Seq data. This is even the case for a unicellular eukaryotic organism like *S. cerevisiae* where the risk of assembling chimeric transfrags is elevated compared to the other organisms chosen in Table 3.1, mainly due to the much higher gene density and frequency of overlapping genes in its genome. Because chimeric events negatively impact the precision performance, a metric BRANCH improves, their frequency is likely to be lower in the transfrags post-processed by BRANCH than the ones from the upstream *de novo* assemblers. It is important to point out here that the current version of BRANCH does not detect or correct chimeric transfrags generated by the *de novo* assemblers. However, future improvements to our software will include such a feature.

(E) Assemblies Assisted with a Related Genome. The sequenc-ing and assembly of a genomic guide sequence can be avoided if a genome from a closely related organism is available, which is an important use case of BRANCH. Table 3.1c gives the test results for such a situation where the genes from rat served as guide sequence for improving the assembly

of RNA-Seq data from mouse. In this data set the sensitivity and precision improves with BRANCH post-processing for Velvet/Oases by 4.4% and 1.1%, and for Trinity by 7.0% and 1.8%, respectively. The other test parameters also show noticeable improvements. The numbers of complete transcripts, complete genes, covered transcripts and proteins annotated in the mouse genome increase by 5.2-8.0%, 6.3-9.6%, 7.0-10.2% and 4.0-6.2%, respectively. Overall the improvements with a closely related genome are slightly less pronounced than with guide contigs from the same organism. This is expected since heterologous sequences represent a more challenging situation where it is important to perform the read and transfrag mapping against the related genome sequences with stringent enough mapping parameters in order to minimize the formation of false positive extension and fusion events of transfrags. When relaxing these parameters one can increase the number of extension events, but often this will result in a decreased precision.

In summary, the above test results demonstrate BRANCH's efficiency in improving the representation of full-length transcripts in *de novo* assemblies by taking advantage of genomic guide sequence information from the same or a closely related organism.

Chapter 4

AlignGraph Algorithm

4.1 Introduction

Previous studies on reference-assisted assemblies include the AMOScmp software (Pop *et al.*, 2004b), an add-on tool for the ARACHNE assembler (Gnerre *et al.*, 2009), and custom workflows largely based on existing assembly software (*e.g.* Schneeberger *et al.*, 2011). The first two were designed primarily for Sanger reads, while the latter has been used for NGS genome assembly. Downstream of the primary assembly, scaffolding algorithms, such as RACA (Kim *et al.*, 2013), can be used that order and orient pre-assembled contigs to a connection map by incorporating additional sequence information from mate pair or paired-end (PE) reads and/or from closely related genomes (Pop *et al.*, 2004c; Boetzer *et al.*, 2011; Dayarian *et al.*, 2010; Gao *et al.*, 2011; Salmela *et al.*, 2011; Gritsenko *et al.*, 2012). The resulting scaffolds contain often gaps, which are unresolved sequence areas between the original contigs. Dedicated gap filling algorithms can be used to partially fill these gaps

(Boetzer and Pirovano, 2012; Luo *et al.*, 2012; Tsai *et al.*, 2010). More recently, components of reference-based strategies have also been incorporated into some of the *de novo* assembly suites themselves such as the *cheat* mode option of ALLPATHS-LG (Gnerre *et al.*, 2011) and IDBA-hybrid (unpublished).

This study proposes a novel algorithm, called AlignGraph, for improving the lengths and completeness of contigs or scaffolds by reassembling them with help provided by a reference genome of a closely related organism. In contrast to existing reference-assisted methods, AlignGraph is a *secondary assembly algorithm* that loads the alignment information of PE reads and pre-assembled contigs/scaffolds against the reference into a novel assembly graph, called the *PE multi-positional de Bruijn graph*, that we specifically designed for facilitating secondary assemblies. By traversing this graph, the contigs or scaffolds of the primary assembly can be extended and joined. AlignGraph differs from most scaffolding algorithms by extending contigs exclusively with resolved rather than unresolved bases (Ns), and by acting either upstream and/or downstream of them.

AlignGraph's functionalities are unique by solving several challenges in improving assembly results. As a de Bruijn graph-based method it solves limitations typical for many heuristic extension methods that are often used in the *de novo* assembly area (Warren *et al.*, 2007a; Jeck *et al.*, 2007; Dohm *et al.*, 2007). For instance, if there are multiple solutions for how to extend a contig, then finding the correct one can be challenging with most heuristic methods. Those ambiguous solutions, which correspond to branched paths in the de Bruijn graph, are usually caused by repetitive sequences in genomes, and frequently lead to early terminations of the contig extension process. The de Bruijn graph method

is often more efficient than heuristic methods in finding the correct solution here, because the contextual information, required for resolving these ambiguities, is maintained in the graph (Zerbino and Birney, 2008a; Chaisson and Pevzner, 2008). This issue is not as pronounced in assemblies with much longer Sanger reads, as those are much more likely to span non-repetitive regions with repetitive regions in between (Gnerre *et al.*, 2009). Thus, it is particularly important to address this problem in assemblies with short reads. In comparison to the conventional de Bruijn graph, our PE multi-positional de Bruijn graph has several additional advantages. First, many branched paths can be eliminated directly in the graph with help of the additional PE read and alignment information. This simplifies the identification of correct paths. Second, many false positive paths, caused by sequencing errors, can be eliminated by correcting erroneous reads with correct reads that align to the same position in the reference genome. Third, guided by the alignment information to the reference genome, the PE multi-positional de Bruijn graph is less affected by regionally low read coverage that often gives rise to incomplete paths in the conventional de Bruijn graph. As a result, many incorrect extensions and early terminations can be avoided.

4.2 Methods

4.2.1 AlignGraph Algorithm

This section describes the AlignGraph algorithm. Its workflow can be divided into the following three major steps. Figure 4.1B illustrates these steps with an example.

(i) *Alignment Maps.* The PE reads are aligned against both the pre-assembled contigs and the close reference genome; and the contigs are aligned against the reference.

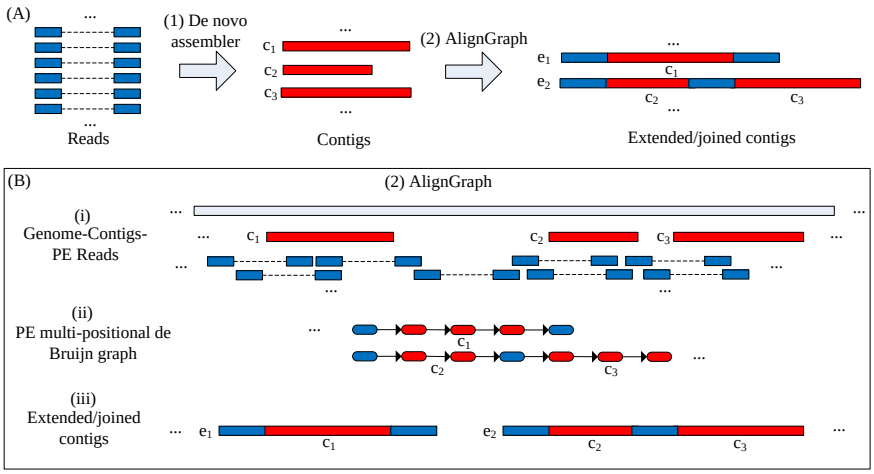


Figure 4.1: Overview of the AlignGraph algorithm. The outline on the top (A) shows AlignGraph in the context of common genome assembly workflows, and the one on the bottom (B) illustrates its three main processing steps. (A) In step 1, the PE reads from a target genome are assembled by a *de novo* assembler into contigs (here c_1, c_2 and c_3). Subsequently (step 2), the contigs can be extended (blue) and joined by AlignGraph (e_1 and e_2). (B) The workflow of AlignGraph consists of three main steps. (i) The PE reads are aligned to the reference genome and to the contigs, and the contigs are also aligned to the reference genome. (ii) The PE multi-positional de Bruijn graph is built from the alignment results, where the red and blue subpaths correspond to the aligned contigs and sequences from PE reads, respectively. (iii) The extended and/or joined contigs (here e_1 and e_2) are generated by traversing the graph.

(ii) *Contig Reassembly.* The alignment mapping results are used to construct a positional variant of the de Bruijn graph, called the *PE multi-positional de Bruijn graph.*

(iii) *Graph Traversal.* The resulting graph is edited and traversed to obtain extended contigs.

Throughout the text, the source genome of the PE reads and the pre-assembled contigs is referred to as the *target genome*, whereas the genome of the closely related species for guiding the contig improvement steps is referred to as the *reference genome.* For simplicity, the following description of AlignGraph refers mostly to contigs, but it also applies to scaffolds containing a limited amount gaps.

Prerequisites. Prior to the above steps, the user is expected to generate genomic PE reads for the target genome of interest and to assemble them with a *de novo* NGS genome assembler. Since most genome assemblers perform better with PE than single end data, AlignGraph also depends on this sequence type. A major advantage of AlignGraph is its design to work with most genome assemblers, but the quality of the initial *de novo* assembled contigs is expected to impact the final results (see 4.3.2). For optimal results, it is also important to follow the recommendations of the chosen *de novo* assembler with respect to insert length of the sequencing library, minimum coverage of the target genome with PE reads and other recommendations. If scaffolds are inputted, it is usually beneficial to fill them with a gap filling algorithm prior to processing them with AlignGraph (*e.g.* Boetzer and Pirovano, 2012). Another requirement for AlignGraph is the availability of a closely related reference genome sequence. Nearly complete reference genomes of high quality will yield the best results, but partially sequenced genomes can be used as well.

(i) Alignment Maps. In the initial preprocessing step of AlignGraph, the PE reads, used for the *de novo* assembly in the *Prerequisite* section, are aligned to the contigs and to the reference genome, and the contigs are also aligned to the reference genome. Aligning the reads to the contigs simplifies their alignments to the reference by guiding them with the much longer contigs as backbone (see below). Generating reliable alignments among the PE reads and the contigs is relatively straightforward, because both are from the same genetic background, thus requiring a low level of variant tolerance in the alignments. Aligning the contigs to the reference genome demands a higher level of variant tolerance. However, due to the relatively large length of the contigs, their alignments to the reference can also be

generated reliably, as long as the evolutionary distance between the target and reference genome is not too large. The current implementation of AlignGraph uses Bowtie2 and BLAT for these two alignment steps, respectively (Langmead and Salzberg, 2012; Kent, 2002). In contrast to this, aligning the relatively short PE reads to the reference genome is a much more challenging task, due to the difficulty of generating reliable short alignments containing larger numbers of mismatches and gaps. This problem does not apply to the reads aligning to the contigs since their alignment positions to the reference genome can be inferred from the more robust contig alignments. For the PE read to reference genome alignment, it is important to choose a highly variant tolerant short read aligner that is able to reliably align most of the short reads to their *true* source locations in the reference genome while minimizing the number of false positive read placements. Clearly, the latter would negatively impact the precision performance of AlignGraph by leading to chimeric joins in the downstream contig extension steps. Although a wide range of short read aligners has been developed over the past years (Li and Homer, 2010), none of them has been specifically designed or optimized for aligning short reads against reference genomes with sequence differences more pronounced than those observed among genomes within the same species. To minimize the above challenges, we have chosen for this critical step the highly tunable Bowtie2 aligner with parameter settings that we optimized for aligning PE reads from a target genome to a reference genome sharing variable degrees of sequence similarity. The use of PE read alignments in this step is also important, because the additional sequence information, provided by the second read in a PE, increases the specificity of the alignment process compared to single end reads, and thus reduces the number of false read placements.

To account for rearrangements among the two genomes, we use for the alignments of the PE reads against the reference genome more relaxed insert length variation settings than in the alignments against the contigs.

(ii) Contig Reassembly with PE Multi-positional de Bruijn Graph. The core functionality of AlignGraph is the extension of the contigs by re-assembling them using the alignment results obtained in the previous step. To achieve this efficiently, we build from the alignment maps a variant of the de Bruijn graph, here called the *paired-end multi-positional de Bruijn graph*. This method combines the PE de Bruijn graph (Medvedev *et al.*, 2011) and the positional de Bruijn graph (Ronen *et al.*, 2012) where we incorporate both PE read information and alignment positions into the graph (Pevzner *et al.*, 2001). The former was designed to generate more complete contigs in *de novo* assemblies, and the latter to correct contig errors in secondary assemblies. Our approach solves several problems in improving assembly results that we briefly discussed in the Introduction (see also Table 4.1). The following describes our modified de Bruijn graph in more details, where we first introduce important concepts of conventional de Bruijn graph-based assembly methods.

Background

The most widely used method for genome assemblies from short reads is the de Bruijn graph method (Pevzner *et al.*, 2001). A de Bruijn graph is a directed graph: two connected nodes represent $k+1$ bases where the first node represents the first k bases and the second node the second k bases (called *k-mer*). To construct a de Bruijn graph, $l-k+1$ connected nodes are constructed from each read of length l and two nodes from different reads are joined if they share the same k-mers. In theory, the graph contains a walk representing

Table 4.1: Problems the PE multi-positional de Bruijn graph solves in comparison to the conventional de Bruijn graph.

Problem	Consequence	Solution
Repeat sequences	Branched paths	Distinguishes paths for repetitive regions by incorporating PE read and alignment position information
Sequencing errors	False positive paths	Corrects paths from erroneous reads with correct reads aligned to the same position
Low sequencing depth	Incomplete paths	Builds paths from reads in low coverage areas supported by reference

the full sequence of the genome if traversed properly. However, this walk is hard to obtain in practice because of false positive, incomplete and branched connections in the graph that are caused by errors in the reads and repeats in the genome. The false positive and incomplete paths are due to false positive k-mers with sequencing errors and missing k-mers from regions of low sequencing depth, respectively. The branched paths are caused by joins of k-mers from repetitive regions. Several variations of the de Bruijn graph have been proposed to solve these limitations, especially the branched paths, while preserving all of its genome information (Medvedev *et al.*, 2011; Ronen *et al.*, 2012; Peng *et al.*, 2010). The paired-end de Bruijn graph (Medvedev *et al.*, 2011) is built from PE reads, where each k-mer contains k bases from the left pair plus its corresponding k bases from the right pair. In contrast to this, the positional de Bruijn graph (Ronen *et al.*, 2012) incorporates read alignment information by including in each k-mer the k bases plus its alignment position. With the additional information assigned to the k-mers, k-mers from repetitive regions can often be distinguished, and thus the number of branches in the graph can be reduced. In addition, because the positional de Bruijn graph is built from read alignments, false positive

and incomplete paths can be largely avoided. We emphasize that the PE de Bruijn graph requires the left pair forward-strand read and the right pair reverse-strand read or vice versa, but it is difficult to know their orientation. This problem can be resolved if the PE de Bruijn graph is built from aligned reads, where their orientation can be obtained from the alignments.

PE Multi-positional de Bruijn Graph

We derive the *paired-end multi-positional de Bruijn graph* as a combination of the PE de Bruijn graph and the positional de Bruijn graph. Each k-mer of the PE multi-positional de Bruijn graph is composed of three left/right element pairs: the k bases of each the left and the right read pair (called *left or right k bases*), the alignment position of each the left and the right k bases to the contigs, and the alignment position of each the left and the right k bases to the reference genome. Two k-mers can be joined if they have similar k bases and close alignment positions within the constraints defined in the formulas below. Formally, let s be the k bases from the left read pair and s' the corresponding k bases from the right read pair, then the k-mer of PE multi-positional de Bruijn graph is a 6-tuple (s, s', c, g, c', g'), where c is the alignment position of s to the contigs, g is the alignment position of s to the reference genome, c' is the alignment position of s' to the contigs, and g' is the alignment position of s' to the reference genome. Two k-mers (s_i, s'_i, c_i, g_i, c'_i, g'_i) and (s_j,

s'_j, c_j, g_j, c'_j, g'_j) can be joined if constrains (1)-(6) are met:

$$\text{mismatch}(s_i, s_j) < \delta \tag{4.1}$$

$$\text{mismatch}(s'_i, s'_j) < \delta \tag{4.2}$$

$$|c_i - c_j| < \epsilon \text{ or } c_i = -1 \text{ or } c_j = -1 \tag{4.3}$$

$$|g_i - g_j| < \epsilon \tag{4.4}$$

$$|c'_i - c'_j| < \epsilon + 2D \text{ or } c'_i = -1 \text{ or } c'_j = -1 \tag{4.5}$$

$$|g'_i - g'_j| < \epsilon + 2D \tag{4.6}$$

where δ and ϵ are small numbers with the default values 5 and 25, respectively, and D is the variability of the insert length I of the PE reads. The variability D is equal to $\max\{I_u - I, I - I_l\}$ where I_u and I_l are the upper and lower limits of I, respectively. The variables in the above formulas are explained below.

δ: To join two k-mers and tolerate sequencing errors, we allow a small number of mismatches δ between s_i and s_j and between s'_i and s'_j in (1) and (2), respectively.

ϵ: We allow a small shift ϵ between each pair of alignment positions in (3)-(6), because the same k bases s_i and s_j (or s'_i and s'_j) from different reads may align to different but close positions in the contigs or genome as discussed in Ronen *et al.* (2012).

$2D$: We allow a shift $2D$ of s'_i and s'_j's alignment positions to the contigs in (5) and to the reference genome in (6). The maximum and minimum alignment distances between a read pair are $I - l + D$ and $I - l - D$, respectively, where l is the read length, assuming the same read length for both members in a pair. Thus, the maximum

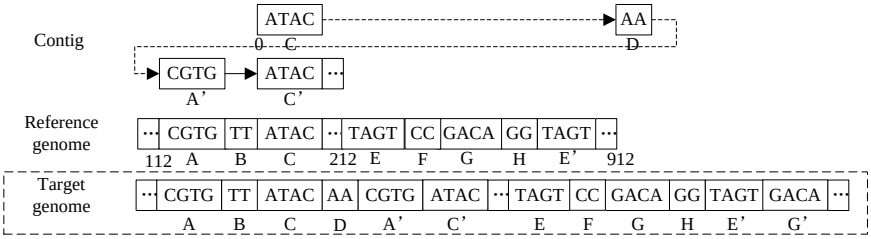
alignment distance of two right reads with left reads aligned at the same position is $(I - l + D) - (I - l - D) = 2D$. This distance is equal to the distance between any two k-mers from the same position in the right read pairs, so the maximum distance between s'_i and s'_j will be $2D$.

-1: s_i and s_j (or s'_i and s'_j) can be joined if one or both of them are aligned directly to the reference genome rather than guided by the *de novo* contigs. In those cases, we assign -1 as alignment position to the contigs. This is important because we allow contig extensions only if the alignable and unalignable bases to contigs can be joined.

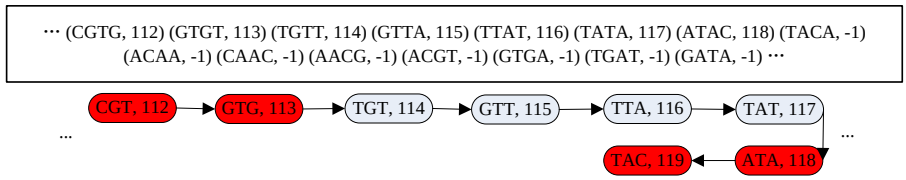
It is important to guarantee that each k-mer corresponding to an insertion of a read alignment has a position in the reference genome. To achieve this, we append the inserted k-mer to the end of the genome sequence. In our implementation of the PE multi-positional de Bruijn graph, we first iteratively load sections of the reference genome into memory. Then we perform the following operation. We test for each k-mer in each aligned read at genome position g, if there is already a k-mer at g and whether the new k-mer can be joined with it. If so then we join the two k-mers; otherwise we attach the new k-mer to position g. The connection between two k-mers is recorded by using pointers and the read coverage for each k-mer is stored along with it. Figure 4.2 illustrates the main advantages of the PE multi-positional de Bruijn graph compared to the positional de Bruijn graph with several examples (see also Table 4.1). This includes the contig-guided PE read alignment against the reference genome resulting in a larger number of alignable reads, and thus a more complete de Bruijn graph (Figure 4.2B); as well as the reduction of branched paths in the graph by distinguishing reads from different repetitive regions (Figures 4.2C and 4.2D). For space

reasons, the advantages over the conventional de Bruijn graph in reducing false positive and incomplete paths are not shown. The PE multi-positional de Bruijn graph can converge to the positional de Bruijn graph by relaxing the above constraints (2), (3), (5) and (6). In our tests, the former shows usually an up to 5% better performance than the latter with respect to several sensitivity measures (see below for definitions). This improvement is considerable because the number of branches that are only resolvable by the PE multi-positional de Bruijn graph is usually not very large.

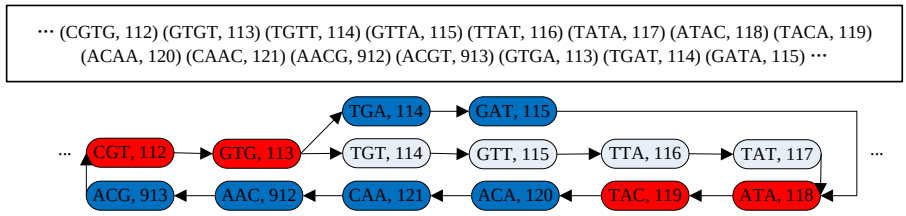
(iii) Graph Traversal Returns Extended Contigs. To remove errors, the de Bruijn graph needs to be edited prior to its traversal. The three major types of errors are tips, bubbles and erroneous connections (Zerbino and Birney, 2008a; Chaisson and Pevzner, 2008; Peng *et al.*, 2010). Most of them are caused by errors in the reads. A tip is a short path with a dead end, while a bubble consists of two short paths sharing the same start and end nodes. Most of the tips and bubbles can be corrected by joining k-mers with $< \delta$ mismatches. The remaining errors can be removed by applying a coverage cut-off filter similar to the strategies employed by most *de novo* assemblers. Due to the additional information encoded in the modified de Bruijn graph, one can use here a relatively small coverage threshold. After these error removal steps, the PE multi-positional de Bruijn graph is traversed, using a breadth-first strategy, to generate the final contigs. Each traversal stops at a branch position and an extended contig is returned. After returning the extended contigs, the remaining unextended contigs (identical with initial *de novo* contigs) are provided to the user in a separate file. Finally, contigs with sufficient PE read connections and a path between them can be joined. Occasionally, those connections can be missed by the above



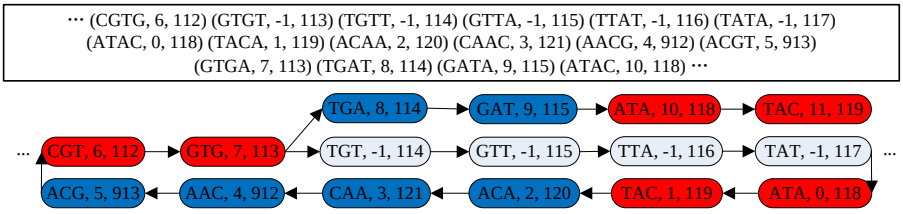
(A) Positional de Bruijn graph



(B) Positional de Bruijn graph using contig to guide read alignment



(C) Multi-positional de Bruijn graph



(D) PE multi-positional de Bruijn graph

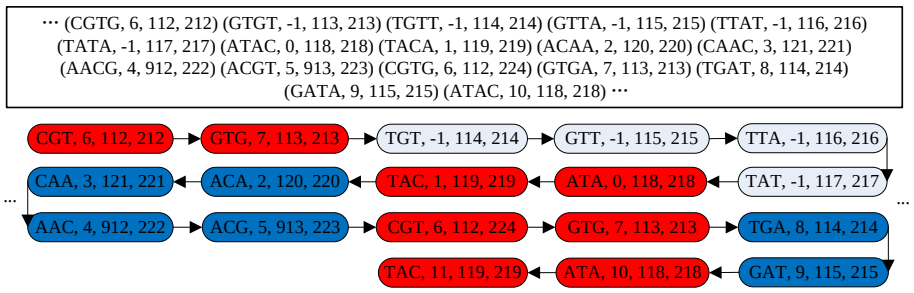


Figure 4.2: (Continued from previous page) Advantages of the PE multi-positional de Bruijn graph compared to the positional de Bruijn graph. In the target genome given on the top A and A', C and C', E and E', G and G' are repetitive regions. Each PE read of length 2×4 bp is sequenced with one pair from region $ABCDA'C'$ and the other from the corresponding position of region $EFGHE'G'$ (the pair from $EFGHE'G'$ is omitted for simplicity). In comparison to the target genome, the reference genome has a repeat-free region ABC similar to $ABCDA'C'$ and a region $EFGHE'$ similar to $EFGHE'G'$. The reads from region $ABCDA'C'$ are assembled with a *de novo* assembler into a contig starting from $CDA'C'$, but regions A and B are not assembled due to low sequencing depth, repeats or other problems. When aligning the contig to the reference genome, the repetitive regions C and C' are both aligned to C in the reference genome and the insertion D is assigned to the end of the reference. In (A) reads are aligned directly to the reference genome to build the initial positional de Bruijn graph; and in (B)-(D) the reads are aligned to the pre-assembled contigs and then aligned to the reference to build first the extended positional de Bruijn graph and then the PE multi-positional de Bruijn graph. (A) The initial positional de Bruijn graph is built here with 3-mers. Some reads cannot be aligned to the reference genome due to sequence differences in the target genome as indicated here by 3-mers with -1 as alignment position. The repetitive regions A and A' (or C and C') are collapsed into one path in red in the graph. (B) The initial positional de Bruijn graph is constructed with help from the read-to-contig alignment information. The read-to-reference genome alignment information yields a more complete positional de Bruijn graph, but the repetitive regions A and A' (or C and C') are still collapsed resulting in branch points. (C) An extended positional de Bruijn graph is built by incorporating into each 3-mer the read alignment position to the contig. As a result of this operation, the repetitive regions C and C' can be distinguished into two paths where the 3-mers have different alignment positions in the contig, but A and A' are still collapsed. (D) The PE multi-positional de Bruijn graph is constructed by incorporating into each 3-mer their PE read alignment positions to the reference genome (the right 3 bases and its alignment position to the contig is omitted here). With this information the repeats A and A' can be distinguished into two paths as the 3-mers have different PE alignment positions in the reference genome. The final graph contains only one single path allowing to output an extended contig corresponding to the region $ABCDA'C'$ in the target genome.

filtering step because of too low read coverage in local areas of the connecting path.

4.2.2 Software Implementation

AlignGraph is implemented in C++ for Linux operating systems. Its required input includes the PE reads, the pre-assembled *de novo* contigs, and the reference genome. Its output includes the extended contigs as well as the remaining non-extended contigs. AlignGraph runs the alignment steps with BLAT and Bowtie2 automatically, but both need to be installed on the system. AlignGraph's run time is currently 23-57 minutes per million aligned reads. In the performance tests of this study the memory usage was 36-50 GB, and it stays below 100 GB even for entire mammalian genomes. These requirements are much more moderate than those of most *de novo* assemblers (Luo *et al.*, 2012).

4.3 Evaluation

4.3.1 Experimental Design

Background. To evaluate AlignGraph's efficiency in improving genome assemblies, we performed a series of systematic performance tests. For this, we downloaded publicly available assemblies and/or assembled genomic PE read sets from organisms of variable genome complexity with seven widely used *de novo* assemblers, extended the resulting contigs with AlignGraph, and then evaluated the improvements with a set of standard metrics for comparing assembly results (Table 4.2). In these tests it was important to choose the NGS read samples from organisms where the genome sequence of both the target genome and a close reference genome are known. This way one can evaluate the completeness and correct-

ness of the results against a true result rather than one that is unknown or only partially known. To also assure the improvements obtained by AlignGraph are not simply the result of insufficient optimizations of the upstream *de novo* assembly, we included pre-assembled contig and scaffold sets that are widely accepted by the community as benchmark data sets for evaluating assembly software. Today's requirements for assembling genomes from NGS were met by choosing read samples with $\geq$75bp and paired-end read information. In total we performed assembly tests on the following three sample sets.

(A) A. thaliana Sample. The first sample set was from the model organism *A. thaliana*, which is a flowering plant with a compact genome of 130Mb in size. The PE read set chosen for this test is from a genomic Illumina NGS library with a read length of 2×75 bp. As *de novo* assemblers, we included in this test Velvet and ABySS, which we chose here as software representatives performing well on single library data, and because of their good sensitivity and precision performance (Lin *et al.*, 2011). The VelvetOptimiser tool was used to optimize the parameter settings for the Velvet assembly. ABySS was run with the same k-mer length as Velvet, while the remaining parameters were set to their defaults. To extend the preassembled contigs with AlignGraph, we used the publicly available genome sequence from the related *A. lyrata* species as reference (Table 4.2a). The latter was chosen because it constitutes a more challenging reference genome for testing AlignGraph's performance in improving genome assemblies than the references used in the other tests below. This is the case for the following reasons (Hu *et al.*, 2011): *A. lyrata* and *A. thaliana* diverged over 10 million years ago; their genomes differ by many regional rearrangements; the sequence similarity in the common regions of their genomes is only 80%; and the *A. lyrata* genome

Table 4.2: Performance Evaluation of AlignGraph. (a) Genomic PE reads from *A. thaliana* were assembled with Velvet and ABySS. The resulting contigs were extended with AlignGraph using as reference the genome sequence from *A. lyrata*. (b-d) The subsequent panels contain assembly results for the human chromosome 14 sample from the GAGE project where the chimpanzee genome served as reference. (b) Contig assembly results are given for the *de novo* assemblers ALLPATHS-LG, ALLPATHS-LGc (in cheat mode), SOAPdenovo, MaSuRCA, CABOG and Bambus2. (c) Scaffolded assembly results are given for SOAPdenovo, MaSuRCA, CABOG and Bambus2. The results are organized row-wise as follows: the number of initial contigs obtained by each *de novo* assembler[1], the 'extendable' subset of the initial contigs that AlignGraph was able to improve[2], and the extension results obtained with AlignGraph[3]. The additional columns give the number of contigs[4], N50 values[5], the number of covered bases[6], the average[7] and maximum[8] length of the contigs, the number of misassemblies per million base pairs (MPMB)[9], and the average identity among the true contigs and the target genome[10]. More details on these performance criteria are provided in 4.3.1(E).

Upstream assembler	Contig set	N Contigs[4]	N50[5]	N Covered bases [6]	Average length[7]	Maximum length[8]	MPMB[9]	Average identity[10]
		(a) Contigs of A. thaliana genome						
Velvet	All[1]	30,037	3,515	82,844,417	2,668	27,792	22.2	95.2%
	Extendable[2]	8,615	4,148	28,007,451	3,262	27,398	0.3	97.6%
	Extendable + AlignGraph[3]	5,751	7,876	32,467,110	5,521	49,768	1.6	94.8%
ABySS	All	30,972	2,559	69,432,667	2,206	29,760	13.4	97.2%
	Extendable	11,693	2,820	28,885,212	2,454	16,343	0.5	98.7%
	Extendable + AlignGraph	8,427	5,484	35,859,786	4,151	25,321	1.1	95.8%

(Continued from previous page)

(b) Contigs of Human chromosome 14								
ALLPATHS-LG	All	4,383	38,590	83,849,397	19,201	240,764	0.3	98.9%
	Extendable	1,674	39,851	35,746,095	20,806	200,495	0.1	98.9%
	Extendable + AlignGraph	785	71,847	36,441,001	45,358	305,880	0.0	97.5%
ALLPATHS-LGc	All	3,856	43,856	83,860,939	21,818	275,446	0.2	99.3%
	Extendable	1,296	45,719	31,457,201	24,346	275,446	0.1	99.5%
	Extendable + AlignGraph	608	86,613	34,614,465	54,406	294,615	0.0	96.9%
SOAPdenovo	All	10,865	16,855	80,135,941	7,623	147,494	5.9	94.9%
	Extendable	5,613	17,412	45,246,077	8,223	141,981	0.9	96.4%
	Extendable + AlignGraph	3,469	32,881	52,861,640	15,271	219,841	0.5	95.0%
MaSuRCA	All	19,034	5,767	75,497,302	3,802	53,837	13.9	98.9%
	Extendable	9,241	6,047	38,842,517	4,199	51,249	0.2	99.2%
	Extendable + AlignGraph	5,665	11,590	43,930,184	7,666	66,758	0.4	98.1%
CABOG	All	3,118	46,523	84,989,190	27,401	296,888	0.3	97.3%
	Extendable	1,692	45,669	46,499,763	27,089	296,888	0.0	98.7%
	Extendable + AlignGraph	701	101,907	50,527,605	70,362	443,952	0.1	97.6%
Bambus2	All	11,219	8,378	64,011,072	5,764	449,449	3.1	89.9%
	Extendable	6,995	7,521	37,857,989	5,439	62,798	0.3	97.6%
	Extendable + AlignGraph	2,722	19,989	39,147,357	14,176	86,154	0.5	96.5%
(c) Scaffolds of Human chromosome 14								
SOAPdenovo	All	3,902	391,693	85,417,248	24,397	1,852,152	1.0	82.9%
	Extendable	901	387,309	40,296,035	47,526	1,019,659	0.1	84.5%
	Extendable + AlignGraph	767	544,209	47,823,279	63,525	2,246,638	0.1	81.0%
MaSuRCA	All	721	580,822	65,433,305	63,876	2,943,966	1.3	57.2%
	Extendable	101	289,703	5,554,781	52,820	1,516,804	0.0	81.9%
	Extendable + AlignGraph	78	316,946	6,986,224	86,552	1,573,741	0.0	83.4%
CABOG	All	471	387,876	81,163,688	176,590	1,944,475	0.1	91.9%
	Extendable	146	358,688	29,372,033	200,539	1,905,529	0.0	98.2%
	Extendable + AlignGraph	67	906,407	33,708,925	481,712	2,051,503	0.0	94.1%
Bambus2	All	569	319,334	64,378,693	116,582	1,477,847	0.1	77.4%
	Extendable	66	272,436	6,949,338	119,858	641,463	0.0	92.0%
	Extendable + AlignGraph	80	377,905	8,963,132	114,852	812,353	0.1	85.4%

sequence is still incomplete and fragmented into many scaffolds.

(B) Human Sample from GAGE. The second sample set is from the community project GAGE (Genome Assembly Gold-standard Evaluations), from which we selected the sample for the human chromosome 14 (Salzberg *et al.*, 2012). Its Illumina sequences consist of PE reads with a length of 76-101 bp from three different libraries. We downloaded the pre-assembled contig sets provided by the GAGE project for the five assemblers that ranked highest in the benchmark tests by Salzberg *et al.* (2012) in assemblies from multiple genome libraries with variable insert lengths. Those included ALLPATHS-LG, SOAPdenovo, MaSuRCA, CABOG and Bambus2. As reference sequence for guiding AlignGraph, we used the chimpanzee genome. For ALLPATH-LG in its *cheat* mode, we reassembled the contigs ourselves, because this assembler exhibits a better sensitivity and precision performance when providing a closely related reference genome. Here it was important to compare the performance of ALLPATHS-LG with AlignGraph when both are guided by the same reference genome.

In addition to contigs, we evaluated AlignGraph's performance in improving the scaffold sets provided by the GAGE project for the same human sample set. Prior to their reassembly with AlignGraph, we reduced the number of unresolved sequence regions (gaps filled with ambiguous N bases) in the scaffolds by applying the GapFiller algorithm, which is currently one of the most efficient gap filling methods (Boetzer and Pirovano, 2012).

To also evaluate the influence of the similarity shared among the reference and target genomes on AlignGraph's performance, we included tests with four reference genomes of variable similarity to the human genome. The reference genomes chosen for this test

were from gorilla, orangutan, gibbon and macaque. The genome sequence from gibbon was the only one that is still incomplete containing scaffolds rather than fully assembled chromosomes.

(C) Published Genome. In addition to the tests above, we were interested in evaluating to what extent AlignGraph can improve the genome sequence generated with another reference assisted assembly approach. For this test, we chose the published genome sequence from Landsberg *erecta* (L*er*-1; Schneeberger *et al.*, 2011). The latter is a strain of *A. thaliana* which is too diverged from the known references to resolve its genome sequence with a simple resequencing approach. Thus, Schneeberger *et al.* (2011) assembled its genome with a reference-assisted pipeline approach that included ALLPATHS-LG and several refinement steps.

(D) Data Sources. The genome sequences used in the above tests were downloaded from the following community sites: *A. thaliana* from TAIR, *A. lyrata* from JGI, Landsberg *erecta* from 1001 Genomes, human and other primates from Ensembl. From the GAGE site, we downloaded the PE read sets, and the pre-assembled contigs and scaffolds for the human chromosome 14 sample (Salzberg *et al.*, 2012). The PE read sets from *A. thaliana* and Landsberg *erecta* were downloaded from NCBI's Sequence Read Archive (SRA) and the 1001 Genome site, respectively. The *A. thaliana* read set contained 32 million 2 $\times$ 75 bp PE reads (accession: SRR073127), the human read set contained 62 million 2 $\times$ 76-101 bp PE reads, and the L*er*-1 read set contained 73 million 2 $\times$ 40-101 bp PE reads.

(E) Performance Measurements. Most of the performance measures used by this study are

adapted from the GAGE project (Salzberg *et al.*, 2012). To evaluate the completeness of the contigs, we aligned them to the target genome with BLAT. If a contig could not be aligned as a single global alignment, it was split according to the local alignment results into the smallest possible number of sub-contigs. The resulting contigs are called *true contigs*. The precision measures include the number of misassemblies per million base pairs (MPMB) and the average identity between contigs and target genome. Misassemblies caused by misjoin errors result in chimeric contigs. Their number can be calculated as the number of splits necessary to obtain the true contigs. Thus, MPMB $= \frac{m}{L} \times 10^6$, where m is the numbers of misassemblies, and L is the cumulative length of the contigs. The average identity between true contigs and the target genome is calculated as $\frac{\sum_n t_i \times l_i}{\sum_n l_i}$ where t_i is the identity for contig i and l_i is the length of contig i $(0 < i \leq n)$. In this formula, the identity t_i of the true contigs i is calculated as the number of aligned bases over the length of the alignment. The sensitivity measures include the N50 value and the number of covered bases. The former is the contig size at 50% of the total number of contig bases, and the latter is the total number of genome bases covered by the contigs. Two additional measures are the average length and maximum length of the true contigs. In all tests, we considered only contigs with a length of at least 1000 bp, but used the entire set (including the shorter ones) in AlignGraph's extension steps.

4.3.2 Results

(A) Extension of A. thaliana Contigs. The performance test results for the *A. thaliana* data set are given in Table 4.2a. In comparison to the initial contig sets assembled by Velvet

or ABySS, AlignGraph extends 28.7-37.8% of them when guided by the *A. lyrata* genome as reference. The resulting set of extended contigs contains 27.9-33.2% less sequences, because AlignGraph has joined many of the initial contigs. This leads to improvements of the N50, the number of covered bases, average contig length and maximum contig length for the extendable contig set by 89.9-94.5%, 15.9-24.1%, 69.3-69.2% and 54.9-81.6%, respectively. These improvements are accompanied only by minor increases of MPMB errors. The MPMB values of the extendable and extended contigs are usually much lower than for the complete set, because of their pre-filtered nature that improves their quality. As expected the average identity also drops slightly (2.8-2.9%), because with increased length of the assembled sequences, internal sequence variations accumulate and complicate the alignment of the extended contigs against the target genome. A similar trend can be seen in the results below for the much longer scaffolds where the average identity is always lower for all of the tested assemblers (Table 4.2c). For all three sample sets (4.3.2(A)-4.3.2(D)), the AlignGraph results contain a comparable number of sequence variations to the target genomes as the results of *de novo* assemblers (data not shown). This indicates a high sequence quality of the reassembled contigs.

(B) Extension of Human Contigs and Scaffolds from GAGE. The test results for the human chromosome 14 contigs are given in Table 4.2b. Of the contigs assembled by ALLPATHS-LG, 38.2% can be extended and the extension result contains 53.1% less contigs due to the joins generated by AlignGraph. These improvements are more pronounced than in the above experiment with *A. lyrata* as reference, because the genomes of human and chimpanzee share a much higher sequence similarity than the genomes of *A. thaliana* and *A.*

lyrata. The N50, the number of covered bases, average contig length and maximum contig length for the extendable contig set consistently improve by 80.3%, 1.9%, 118.0% and 52.6%, respectively. Similar results could be obtained with the other *de novo* assemblers SOAPdenovo, MaSuRCA, CABOG and Bambus2. After AlignGraph processing their extendable contigs improved for the same four evaluation metrics by 88.8-165.8%, 3.4-16.8%, 82.6-160.6% and 30.3-54.8%, respectively. If ALLPATHS-LG is run in its cheat mode by guiding it with the same reference genome as AlignGraph, then both the sensitivity and precision measures of the ALLPATHS-LGc contigs improve compared to the assembly without a reference. Nevertheless, AlignGraph is still able to extend 33.6% of the ALLPATHS-LGc contigs and the extension results contain 53.1% less contigs, while the four evaluation metrics improve by 89.4%, 10.0%, 123.5% and 7.0%, respectively. These improvements indicate that the reference-assisted approach used by AlignGraph is more efficient than the one from ALLPATHS-LG in its cheat mode at the contig assembly stage.

AlignGraph's performance results on the scaffolds from the same human chromosome 14 dataset are given in Table 4.2c. The scaffold sets from SOAPdenovo, MaSuRCA, CABOG and Bambus2 contain much smaller numbers of sequences than their corresponding contig sets. Nevertheless, AlignGraph is able to extend 11.6-31.0% of them and improve the N50 value and the number of covered bases by 9.4-152.7% and 14.8-29.0%, respectively. The extension results for the scaffold set of Bambus2 contain a slightly larger number of sequences (14) than the extendable set. The reason for this is that many of them are very short and AlignGraph extends them often to scaffolds with a length above the 1000 bp requirement, thus increasing the number of reported scaffolds (see Section 3.1.5). This trend

also explains the slightly lower average length of the extended scaffold set from Bambus2.

(C) Influence of Similarity of Reference Genome. To assess AlignGraph's performance with reference genomes of variable similarity to the target genome, we post-processed the *de novo* assemblies of the human data set with AlignGraph using as reference the genome sequences from five different primates. The columns in Table 4.3 list these organisms according to their evolutionary distances to the human genome (increasing from left to right). To avoid confusions,o exact sequence similarity values to the human genome are not provided because there are many possibilities how to calculate them which can lead to very different results. As expected the performance measures degrade with the evolutionary distance between the target and reference genomes. While the first four reference genomes show respectable improvements, the macaque genome seems to be too diverged from human to achieve any major improvements. However, this performance drop is mainly due to the difficulty of aligning short reads to a highly diverged reference. Future improvements in NGS read length and alignment algorithms are likely to enhance AlignGraph's performance in this regard.

(D) Improvements to Published Genome. The test results for the published Landsberg *erecta* genome are shown in Table 4.4.The initial scaffold set used in this test consisted of 1,676 sequences. AlignGraph extended 27.6% of these scaffolds, while the extended set contains 20.3% fewer scaffolds. In addition, AlignGraph improves the N50 value, the number of covered bases, the average contig length and maximum length values for the extendable scaffolds by 86.6%, 8.1%, 35.7% and 8.1%, respectively. These improvements demonstrate AlignGraph's usefulness in improving published genome sequences, even for those that have

Table 4.3: Performance with reference genomes of variable similarity. The tests were performed on the human chromosome 14 sample where the listed primate genomes served as reference. The results include the percentage values of alignable reads[1], extendable contigs relative to the initial set[2] and improvements of the N50 values relative to the extendable contigs[3]. Due to space limitations, the latter two rows contain averaged percentage values for the five assemblers ALLPATHS-LG, SOAPdenovo, MaSuRCA, CABOG and Bambus2.

Percentage of	Chimpanzee	Gorilla	Orangutan	Gibbon	Macaque
Aligned reads[1]	94.5%	91.6%	88.9%	49.9%	24.9%
Extendable contigs[2]	51.0%	36.4%	24.9%	6.7%	-
Improved N50[3]	109.9%	84.0%	73.2%	65.3%	-

Table 4.4: Improvements to Published Genome. The published scaffolds from Landsberg *erecta* were extended with AlignGraph using the *A. thaliana* genome as reference. The rows and columns are arranged the same way as in Table 4.2, but several columns are missing here, because it is not possible to compute the corresponding performance measures in a meaningful manner without having access to a 'true' target genome sequence. In addition, we report here the total number of bases in the contigs[1].

Contig set	N Contigs	N50	N total bases[1]	Average length	Maximum length
All	1,676	341,653	112,578,343	67,170	2,930,180
Extendable	462	448,682	57,574,961	124,621	2,930,180
Extendable+AlignGraph	368	837,458	62,216,675	169,067	3,168,537

been carefully curated by their authors.

In summary, the above performance tests demonstrate AlignGraph's efficiency in improving the results of a variety of *de novo* assemblers and species with variable genome complexity by taking advantage of closely related reference genomes.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

SEED is an efficient method for clustering very large NGS data sets while allowing up to three mismatches and three overhanging residues to their virtual center. The method gains its performance from a block spaced seed method that greatly accelerates the downstream clustering process. With increasing numbers of sequences the method shows a linear time and memory performance. It is able to cluster on a single CPU core 100 million sequences in less than four hours, while using not more than 8 GB of memory. These are very reasonable resource requirements for modern computers. The current implementation of SEED is optimized to handle sequences of 21-200 bp in length. This matches at the moment the length range of most of the widely used NGS technologies, such as Illumina's reversible terminator method. As preprocessing and data reduction tool, SEED is very efficient in improving the time and memory requirements of downstream NGS data processing routines,

such as genome and transcriptome assemblies, often by a factor of 2- to 5-fold, based on the NGS test data sets used in this study. Moreover, reducing the redundancies in NGS data with SEED does not negatively impact the quality of the contigs in downstream assembly steps. In case of the Velvet/Oasis assembler, the N50 values of transcriptome and genome assemblies could be improved with SEED preprocessing by 12-27%.

BRANCH is an efficient reference assisted post-processing method for enhancing *de novo* transcriptome assemblies. It can be used in combination with most *de novo* transcriptome assembly software tools. The assembly improvements are achieved with help from partial or complete genomic sequence information. They can be obtained by sequencing and assembling a genomic DNA sample in addition to the RNA samples required for a transcriptome assembly project. This approach is practical because it requires only preliminary genome assembly results in form of contigs. Nowadays, the latter can be generated with very reasonable cost and time investments. In case the genome sequence of a closely related organism is available, one can skip the genome assembly step and use the related gene sequences instead.

AlignGraph is a novel de Bruijn graph-based algorithm for improving *de novo* genome assemblies guided by sequence information from a closely related species. The chosen PE multi-positional de Bruijn graph approach provides an elegant and efficient solution to this problem. Our performance results demonstrate that the implemented AlignGraph software is able to improve the results of a wide range of *de novo* assemblers for complex genomes even with relatively diverse and suboptimal guide sequences as reference. Moreover, our results demonstrate AlignGraph's usefulness for improving unfinished genome

assemblies. Another advantage is that AlignGraph can be used in combination with most existing *de novo* assemblers.

5.2 Future Work

The three algorithms SEED, BRANCH and AlignGraph can be further improved to achieve better accuracy, time and memory performance. For example, it will be relatively straightforward to parallelize all three algorithms. In addition, their default parameter settings can be further optimized to achieve better performance. For instance, machine learning techniques can be used for dynamic parameter tuning. The following discusses some specific extensions of each algorithm.

For SEED, the current read length limitation will be elevated to keep up with improvements of NGS technologies. This can be achieved by redesigning a set of spaced seeds of larger weight and length. Another extension will support clustering of PE reads. Here one can cluster PE reads simply by the partitioning results obtained from only one read pair, while the other read pair will be placed into the same clusters. The other option is to check both read pairs and put PE reads into one cluster if both of their left read pairs and right read pairs meet the similarity requirements. For BRANCH and AlignGraph, as misassembly detection and correction tool will be added. This can be achieved by techniques similar to those proposed by Gnerre *et al.* (2011). Their main idea is to determine if a transfrag or contig can be aligned to more than one location in the reference genome.

Bibliography

Au, K., Jiang, H., Lin, L., Xing, Y., and Wong, W. (2010). Detection of splice junctions from paired-end rna-seq data by splicemap. *Nucleic Acids Research*, **38**(14), 4570–4578.

Bao, E., Jiang, T., Kaloshian, I., and Girke, T. (2011). Seed: efficient clustering of next-generation sequences. *Bioinformatics*, **27**(18), 2502–2509.

Bao, E., Jiang, T., and Girke, T. (2013). Branch: boosting rna-seq assemblies with partial or related genomic sequences. *Bioinformatics*, **29**(10), 1250–1259.

Birney, E. (2011). Assemblies: the good, the bad, the ugly. *Nat Methods*, **8**(1), 59–60.

Boetzer, M. and Pirovano, W. (2012). Toward almost closed genomes with gapfiller. *Genome biology*, **13**(6), R56.

Boetzer, M., Henkel, C. V., Jansen, H. J., Butler, D., and Pirovano, W. (2011). Scaffolding pre-assembled contigs using sspace. *Bioinformatics*, **27**(4), 578–579.

Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I., Belmonte, M., Lander, E., Nusbaum, C., and Jaffe, D. (2008). Allpaths: de novo assembly of whole-genome shotgun microreads. *Genome Research*, **18**(5), 810–820.

Chaisson, M. J. and Pevzner, P. A. (2008). Short read fragment assembly of bacterial genomes. *Genome research*, **18**(2), 324–330.

Cock, P. J., Fields, C. J., Goto, N., Heuer, M. L., and Rice, P. M. (2010). The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic Acids Res*, **38**(6), 1767–1771.

Dayarian, A., Michael, T. P., and Sengupta, A. M. (2010). Sopra: Scaffolding algorithm for paired reads via statistical optimization. *BMC bioinformatics*, **11**(1), 345.

Dezso, B., Jüttner, A., and Kovács, P. (2011). Lemon-an open source c++ graph template library. *Electronic Notes in Theoretical Computer Science*, **264**(5), 23–45.

Dohm, J. C., Lottaz, C., Borodina, T., and Himmelbauer, H. (2007). Sharcgs, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome research*, **17**(11), 1697–1706.

Edgar, R. C. (2010). Search and clustering orders of magnitude faster than BLAST. *Bioinformatics*, **26**(19), 2460–2461.

Feng, J., Li, W., and Jiang, T. (2010). Inference of isoforms from short sequence reads. In *Research in Computational Molecular Biology*, pages 138–157. Springer.

Feng, J., Li, W., and Jiang, T. (2011). Inference of isoforms from short sequence reads. *Journal of Computational Biology*, **18**(3), 305–321.

Ferragina, P. and Manzini, G. (2000). Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE.

Fritz, M. H. Y., Leinonen, R., Cochrane, G., and Birney, E. (2011). Efficient storage of high throughput sequencing data using reference-based compression. *Genome Research*.

Gao, S., Sung, W.-K., and Nagarajan, N. (2011). Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *Journal of Computational Biology*, **18**(11), 1681–1691.

Gnerre, S., Lander, E. S., Lindblad-Toh, K., Jaffe, D. B., *et al.* (2009). Assisted assembly: how to improve a de novo genome assembly by using related species. *Genome Biol*, **10**(8), R88.

Gnerre, S., MacCallum, I., Przybylski, D., Ribeiro, F. J., Burton, J. N., Walker, B. J., Sharpe, T., Hall, G., Shea, T. P., Sykes, S., *et al.* (2011). High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, **108**(4), 1513–1518.

Grabherr, M. G., Haas, B. J., Yassour, M., Levin, J. Z., Thompson, D. A., Amit, I., Adiconis, X., Fan, L., Raychowdhury, R., Zeng, Q., *et al.* (2011). Full-length transcriptome assembly from rna-seq data without a reference genome. *Nature biotechnology*, **29**(7), 644–652.

Grant, G., Farkas, M., Pizarro, A., Lahens, N., Schug, J., Brunk, B., Stoeckert, C., Hogenesch, J., and Pierce, E. (2011). Comparative analysis of rna-seq alignment algorithms and the rna-seq unified mapper (rum). *Bioinformatics*, **27**(18), 2518–2528.

Gritsenko, A. A., Nijkamp, J. F., Reinders, M. J., and de Ridder, D. (2012). Grass: a generic algorithm for scaffolding next-generation sequencing assemblies. *Bioinformatics*, **28**(11), 1429–1437.

Guttman, M., Garber, M., Levin, J. Z., Donaghey, J., Robinson, J., Adiconis, X., Fan, L., Koziol, M. J., Gnirke, A., Nusbaum, C., *et al.* (2010). Ab initio reconstruction of cell type-specific transcriptomes in mouse reveals the conserved multi-exonic structure of lincrnas. *Nature biotechnology*, **28**(5), 503–510.

Hazelhurst, S., Hide, W., Lipták, Z., Nogueira, R., and Starfield, R. (2008). An overview of the wcd EST clustering tool. *Bioinformatics*, **24**(13), 1542.

Hernandez, D., François, P., Farinelli, L., Østerås, M., and Schrenzel, J. (2008). De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. *Genome research*, **18**(5), 802–809.

Hillier, L. W., Reinke, V., Green, P., Hirst, M., Marra, M. A., and Waterston, R. H. (2009). Massively parallel sequencing of the polyadenylated transcriptome of c. elegans. *Genome research*, **19**(4), 657–666.

Holt, R. A. and Jones, S. J. (2008). The new paradigm of flow cell sequencing. *Genome Res*, **18**(6), 839–846.

Hsieh, L. C., Lin, S. I., Shih, A. C., Chen, J. W., Lin, W. Y., Tseng, C. Y., Li, W. H., and Chiou, T. J. (2009). Uncovering small RNA-mediated responses to phosphate deficiency in Arabidopsis by deep sequencing. *Plant Physiol*, **151**(4), 2120–2132.

Hu, T. T., Pattyn, P., Bakker, E. G., Cao, J., Cheng, J. F., Clark, R. M., Fahlgren, N., Fawcett, J. A., Grimwood, J., Gundlach, H., Haberer, G., Hollister, J. D., Ossowski, S., Ottilar, R. P., Salamov, A. A., Schneeberger, K., Spannagl, M., Wang, X., Yang, L., Nasrallah, M. E., Bergelson, J., Carrington, J. C., Gaut, B. S., Schmutz, J., Mayer, K. F., Van de Peer, Y., Grigoriev, I. V., Nordborg, M., Weigel, D., and Guo, Y. L. (2011). The arabidopsis lyrata genome sequence and the basis of rapid genome size change. *Nat Genet*, **43**(5), 476–481.

Huang, X. and Madan, A. (1999). CAP3: A DNA sequence assembly program. *Genome research*, **9**(9), 868.

Idury, R. M. and Waterman, M. S. (1995). A new algorithm for dna sequence assembly. *Journal of Computational Biology*, **2**(2), 291–306.

Jeck, W. R., Reinhardt, J. A., Baltrus, D. A., Hickenbotham, M. T., Magrini, V., Mardis, E. R., Dangl, J. L., and Jones, C. D. (2007). Extending assembly of short dna sequences to handle error. *Bioinformatics*, **23**(21), 2942–2944.

Jiang, H. and Wong, W. (2008). Seqmap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, **24**(20), 2395.

Jiao, Y. and Meyerowitz, E. M. (2010). Cell-type specific analysis of translating RNAs in developing flowers reveals new levels of control. *Mol Syst Biol*, **6**, 419–419.

Johnson, C., Kasprzewska, A., Tennessen, K., Fernandes, J., Nan, G. L., Walbot, V., Sundaresan, V., Vance, V., and Bowman, L. H. (2009). Clusters and superclusters of phased small RNAs in the developing inflorescence of rice. *Genome Res*, **19**(8), 1429–1440.

Kaufmann, K., Wellmer, F., Muiño, J. M., Ferrier, T., Wuest, S. E., Kumar, V., Serrano-Mislata, A., Madueño, F., Krajewski, P., Meyerowitz, E. M., Angenent, G. C., and Riechmann, J. L. (2010). Orchestration of floral initiation by APETALA1. *Science*, **328**(5974), 85–89.

Kececioglu, J. D. and Myers, E. W. (1995). Combinatorial algorithms for dna sequence assembly. *Algorithmica*, **13**(1-2), 7–51.

Kent, W. (2002). Blatthe blast-like alignment tool. *Genome research*, **12**(4), 656–664.

Kim, J., Larkin, D. M., Cai, Q., Zhang, Y., Ge, R.-L., Auvil, L., Capitanu, B., Zhang, G., Lewin, H. A., Ma, J., *et al.* (2013). Reference-assisted chromosome assembly. *Proceedings of the National Academy of Sciences*, **110**(5), 1785–1790.

Langmead, B. and Salzberg, S. L. (2012). Fast gapped-read alignment with bowtie 2. *Nature methods*, **9**(4), 357–359.

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. L. (2009a). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, **10**(3).

Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. (2009b). Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol*, **10**(3), R25.

Leinonen, R., Akhtar, R., Birney, E., Bower, L., Cerdeno-Tárraga, A., Cheng, Y., Cleland, I., Faruque, N., Goodgame, N., Gibson, R., Hoad, G., Jang, M., Pakseresht, N., Plaister, S., Radhakrishnan, R., Reddy, K., Sobhany, S., Ten Hoopen, P., Vaughan, R., Zalunin, V., and Cochrane, G. (2010). The European Nucleotide Archive. *Nucleic Acids Res*.

Levin, J. Z., Yassour, M., Adiconis, X., Nusbaum, C., Thompson, D. A., Friedman, N., Gnirke, A., and Regev, A. (2010). Comprehensive comparative analysis of strand-specific rna sequencing methods. *Nature methods*, **7**(9), 709–715.

Li, H. and Durbin, R. (2009a). Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, **25**(14), 1754–1760.

Li, H. and Durbin, R. (2009b). Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**(14), 1754–1760.

Li, H. and Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, **11**(5), 473–483.

Li, H., Ruan, J., and Durbin, R. (2008). Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome research*, **18**(11), 1851.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and 1000 Genome Project Data Processing Subgroup (2009). The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, **25**(16), 2078–2079.

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

Li, W. and Godzik, A. (2006). Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, **22**(13), 1658–1659.

Li, W., Feng, J., and Jiang, T. (2011). Isolasso: a lasso regression approach to rna-seq based transcriptome assembly. *Journal of Computational Biology*, **18**(11), 1693–1707.

Lin, H., Zhang, Z., Zhang, M. Q., Ma, B., and Li, M. (2008). ZOOM! Zillions of oligos mapped. *Bioinformatics*, **24**(21), 2431–2437.

Lin, Y., Li, J., Shen, H., Zhang, L., Papasian, C. J., *et al.* (2011). Comparative studies of de novo assembly tools for next-generation sequencing technologies. *Bioinformatics*, **27**(15), 2031–2037.

Lindblad-Toh, K., Garber, M., Zuk, O., Lin, M. F., Parker, B. J., Washietl, S., Kheradpour, P., Ernst, J., Jordan, G., Mauceli, E., Ward, L. D., Lowe, C. B., Holloway, A. K., Clamp, M., Gnerre, S., Alföldi, J., Beal, K., Chang, J., Clawson, H., Cuff, J., Di Palma, F., Fitzgerald, S., Flicek, P., Guttman, M., Hubisz, M. J., Jaffe, D. B., Jungreis, I., Kent, W. J., Kostka, D., Lara, M., Martins, A. L., Massingham, T., Moltke, I., Raney, B. J., Rasmussen, M. D., Robinson, J., Stark, A., Vilella, A. J., Wen, J., Xie, X., Zody, M. C., Broad Institute Sequencing Platform and Whole Genome Assembly Team, Baldwin, J., Bloom, T., Chin, C. W., Heiman, D., Nicol, R., Nusbaum, C., Young, S., Wilkinson, J., Worley, K. C., Kovar, C. L., Muzny, D. M., Gibbs, R. A., Baylor College of Medicine Human Genome Sequencing Center Sequencing Team, Cree, A., Dihn, H. H., Fowler, G., Jhangiani, S., Joshi, V., Lee, S., Lewis, L. R., Nazareth, L. V., Okwuonu, G., Santibanez, J., Warren, W. C., Mardis, E. R., Weinstock, G. M., Wilson, R. K., Genome Institute at Washington University, Delehaunty, K., Dooling, D., Fronik, C., Fulton, L., Fulton, B., Graves, T., Minx, P., Sodergren, E., Birney, E., Margulies, E. H., Herrero, J., Green, E. D., Haussler, D., Siepel, A., Goldman, N., Pollard, K. S., Pedersen, J. S., Lander, E. S., and Kellis, M. (2011). A high-resolution map of human evolutionary constraint using 29 mammals. *Nature*, **478**(7370), 476–482.

Luo, R., Liu, B., Xie, Y., Li, Z., Huang, W., Yuan, J., He, G., Chen, Y., Pan, Q., Liu, Y., *et al.* (2012). Soapdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, **1**(1), 18.

Ma, B., Tromp, J., and Li, M. (2002). PatternHunter: faster and more sensitive homology search. *Bioinformatics*, **18**(3), 440–445.

Martin, J. A. and Wang, Z. (2011). Next-generation transcriptome assembly. *Nature Reviews Genetics*, **12**(10), 671–682.

Medini, D., Serruto, D., Parkhill, J., Relman, D. A., Donati, C., Moxon, R., Falkow, S., and Rappuoli, R. (2008). Microbiology in the post-genomic era. *Nat Rev Microbiol*, **6**(6), 419–430.

Medvedev, P., Pham, S., Chaisson, M., Tesler, G., and Pevzner, P. (2011). Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology*, **18**(11), 1625–1634.

Miller, J. R., Delcher, A. L., Koren, S., Venter, E., Walenz, B. P., Brownley, A., Johnson, J., Li, K., Mobarry, C., and Sutton, G. (2008). Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, **24**(24), 2818–2824.

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

Montgomery, T. A., Yoo, S. J., Fahlgren, N., Gilbert, S. D., Howell, M. D., Sullivan, C. M., Alexander, A., Nguyen, G., Allen, E., Ahn, J. H., and Carrington, J. C. (2008). AGO1-miR173 complex initiates phased siRNA formation in plants. *Proc Natl Acad Sci U S A*, **105**(51), 20055–20062.

Myers, E. W. (2005). The fragment assembly string graph. *Bioinformatics*, **21**(suppl 2), ii79–ii85.

Peng, Y., Leung, H. C., Yiu, S.-M., and Chin, F. Y. (2010). Idba–a practical iterative de bruijn graph de novo assembler. In *Research in Computational Molecular Biology*, pages 426–440. Springer.

Peng, Y., Leung, H. C., Yiu, S.-M., and Chin, F. Y. (2011). T-idba: a de novo iterative de bruijn graph assembler for transcriptome. In *Research in Computational Molecular Biology*, pages 337–338. Springer.

Pevzner, P. A., Tang, H., and Waterman, M. S. (2001). An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, **98**(17), 9748–9753.

Phillippy, A. M., Schatz, M. C., and Pop, M. (2008). Genome assembly forensics: finding the elusive mis-assembly. *Genome Biol*, **9**(3).

Picardi, E., Mignone, F., and Pesole, G. (2009). EasyCluster: a fast and efficient gene-oriented clustering tool for large-scale transcriptome data. *BMC bioinformatics*, **10**(Suppl 6), S10.

Pop, M., Phillippy, A., Delcher, A. L., and Salzberg, S. L. (2004a). Comparative genome assembly. *Brief Bioinform*, **5**(3), 237–248.

Pop, M., Phillippy, A., Delcher, A. L., and Salzberg, S. L. (2004b). Comparative genome assembly. *Briefings in bioinformatics*, **5**(3), 237–248.

Pop, M., Kosack, D. S., and Salzberg, S. L. (2004c). Hierarchical scaffolding with bambus. *Genome research*, **14**(1), 149–159.

Qu, W., Hashimoto, S., and Morishita, S. (2009). Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing. *Genome Res*, **19**(7), 1309–1315.

Rao, D., Moler, J., Ozden, M., Zhang, Y., Liang, C., and Karro, J. (2010). PEACE: Parallel Environment for Assembly and Clustering of Gene Expression. *Nucleic acids research*, **38**(suppl 2), W737.

Rasmussen, K. R., Stoye, J., and Myers, E. W. (2006). Efficient q-gram filters for finding all ε-matches over a given length. *Journal of Computational Biology*, **13**(2), 296–308.

Robertson, G., Schein, J., Chiu, R., Corbett, R., Field, M., Jackman, S. D., Mungall, K., Lee, S., Okada, H. M., Qian, J. Q., *et al.* (2010). De novo assembly and analysis of rna-seq data. *Nature methods*, **7**(11), 909–912.

Ronen, R., Boucher, C., Chitsaz, H., and Pevzner, P. (2012). Sequel: improving the accuracy of genome assemblies. *Bioinformatics*, **28**(12), i188–i196.

Salmela, L., Mäkinen, V., Välimäki, N., Ylinen, J., and Ukkonen, E. (2011). Fast scaffolding with small independent mixed integer programs. *Bioinformatics*, **27**(23), 3259–3265.

Salzberg, S. L., Phillippy, A. M., Zimin, A., Puiu, D., Magoc, T., Koren, S., Treangen, T. J., Schatz, M. C., Delcher, A. L., Roberts, M., *et al.* (2012). Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome Research*, **22**(3), 557–567.

Schatz, M. C., Phillippy, A. M., Sommer, D. D., Delcher, A. L., Puiu, D., Narzisi, G., Salzberg, S. L., and Pop, M. (2013). Hawkeye and AMOS: visualizing and assessing the quality of genome assemblies. *Brief Bioinform*, **14**(2), 213–224.

Schmidt, B., Sinha, R., Beresford-Smith, B., and Puglisi, S. J. (2009). A fast hybrid short read fragment assembly algorithm. *Bioinformatics*, **25**(17), 2279–2280.

Schneeberger, K., Ossowski, S., Ott, F., Klein, J. D., Wang, X., Lanz, C., Smith, L. M., Cao, J., Fitz, J., Warthmann, N., *et al.* (2011). Reference-guided assembly of four diverse arabidopsis thaliana genomes. *Proceedings of the National Academy of Sciences*, **108**(25), 10249–10254.

Schulz, M. H., Zerbino, D. R., Vingron, M., and Birney, E. (2012). Oases: robust de novo rna-seq assembly across the dynamic range of expression levels. *Bioinformatics*, **28**(8), 1086–1092.

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

Simpson, J., Wong, K., Jackman, S., Schein, J., Jones, S., and Birol, İ. (2009). Abyss: a parallel assembler for short read sequence data. *Genome research*, **19**(6), 1117–1123.

Simpson, J. T. and Durbin, R. (2010). Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, **26**(12), i367–i373.

Simpson, J. T. and Durbin, R. (2012). Efficient de novo assembly of large genomes using compressed data structures. *Genome Research*, **22**(3), 549–556.

Trapnell, C., Pachter, L., and Salzberg, S. (2009). Tophat: discovering splice junctions with rna-seq. *Bioinformatics*, **25**(9), 1105–1111.

Trapnell, C., Williams, B. A., Pertea, G., Mortazavi, A., Kwan, G., van Baren, M. J., Salzberg, S. L., Wold, B. J., and Pachter, L. (2010). Transcript assembly and quantification by rna-seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nature biotechnology*, **28**(5), 511–515.

Tsai, I. J., Otto, T. D., and Berriman, M. (2010). Method improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps.

Wang, E., Sandberg, R., Luo, S., Khrebtukova, I., Zhang, L., Mayr, C., Kingsmore, S., Schroth, G., and Burge, C. (2008). Alternative isoform regulation in human tissue transcriptomes. *Nature*, **456**(7221), 470–476.

Wang, K., Singh, D., Zeng, Z., Coleman, S., Huang, Y., Savich, G., He, X., Mieczkowski, P., Grimm, S., Perou, C., *et al.* (2010). Mapsplice: accurate mapping of rna-seq reads for splice junction discovery. *Nucleic acids research*, **38**(18), e178.

Warren, R. L., Sutton, G. G., Jones, S. J., and Holt, R. A. (2007a). Assembling millions of short dna sequences using ssake. *Bioinformatics*, **23**(4), 500–501.

Warren, R. L., Sutton, G. G., Jones, S. J., and Holt, R. A. (2007b). Assembling millions of short DNA sequences using SSAKE. *Bioinformatics*, **23**(4), 500–501.

Weber, K. P., De, S., Kozarewa, I., Turner, D. J., Babu, M. M., and de Bono, M. (2010). Whole genome sequencing highlights genetic changes associated with laboratory domestication of c. elegans. *PLoS One*, **5**(11), e13922.

Wu, T. and Watanabe, C. (2005). Gmap: a genomic mapping and alignment program for mrna and est sequences. *Bioinformatics*, **21**(9), 1859.

Xie, Y., Wu, G., Tang, J., Luo, R., Patterson, J., Liu, S., Huang, W., He, G., Gu, S., Li, S., *et al.* (2013). Soapdenovo-trans: De novo transcriptome assembly with short rna-seq reads. *arXiv preprint arXiv:1305.6760*.

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

Zerbino, D. R. and Birney, E. (2008b). Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res*, **18**(5), 821–829.

Zhang, W., Chen, J., Yang, Y., Tang, Y., Shang, J., and Shen, B. (2011). A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies. *PLoS One*, **6**(3).

Zhao, Q., Wang, Y., Kong, Y., Luo, D., Hao, P., and Li, X. (2011). Optimizing de novo transcriptome assembly from short-read rna-seq data: a comparative study. *BMC Bioinformatics*, **12**(Suppl 14), S2.

Zimin, A. V., Marçais, G., Puiu, D., Roberts, M., Salzberg, S. L., and Yorke, J. A. (2013). The masurca genome assembler. *Bioinformatics*, **29**(21), 2669–2677.

Appendix A

Supplementary Materials for Chapter 2

Proof of Theorem 1

Proof. Note that $\frac{l}{gcd(l,w)}$ is the number of blocks in a seed and $\frac{w}{gcd(l,w)}$ the number of blocks of 1's. Hence, $\frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$ is the number of blocks of 0's. If $k > \frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$, then for any set of block spaced seeds with length l and weight w, we can easily find two sequences of length l that differ by k mismatches located in k different blocks. These two sequences will not share a bucket in any hash table since at least one of the k blocks is of 1's, and therefore the block spaced seed set does not achieve full sensitivity.

If $k \leq \frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$, we construct a set of block spaced seeds by including all block spaced seeds whose first $\frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)} - k$ blocks are filled with 0's and other blocks form an arbitrary permutation of k blocks of 0's and $\frac{w}{gcd(l,w)}$ blocks of 1's. The number of

such block spaced seeds is $\binom{\frac{w}{gcd(l,w)}+k}{k}$. For any k mismatches between two sequences with length l, there always exists a block spaced seed having k blocks of 0's where the mismatches could be located in. Therefore, it is easy to verify that these seeds achieve full sensitivity with respect to k mismatches. □

Analysis of Algorithm 1

As suggested by Lin *et al.* (2008), to be time efficient, the size c of the block spaced seed set should be made as small as possible, whereas the weight w of the block spaced seeds should be maximized while satisfying the memory constraint. We require $w \leq 13$, because the headers of our hash tables take too much memory with larger values. For instance, when $w = 14$, they will require $2 \times 4^{14+1} = 2GB$ memory per spaced seed according to Equation (1). Our experience also shows that w should be generally at least 11 in order to achieve a decent time efficiency. Algorithm 1 computes the number of block spaced seeds required to guarantee full sensitivity and to minimize the memory usage of the hash table headers for any given read length l and similarity threshold k. It searches for the best weight w between 11 and 13, and for each weight w, it uses Theorem 1 to obtain an upper bound on the minimum number of required block seeds as $\binom{\frac{w}{gcd(l,w)}+k}{k}$, whenever $k > \frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)}$ holds. When there exist several weights w that yield the same minimum header memory usage, it chooses the largest w which implies the best time efficiency. After w is chosen, a set of block spaced seeds is generated according to the proof of Theorem 1, by including all block spaced seeds whose first $\frac{l}{gcd(l,w)} - \frac{w}{gcd(l,w)} - k$ blocks are filled with 0's and other blocks form an arbitrary permutation of k blocks of 0's and $\frac{w}{gcd(l,w)}$ blocks of 1's.

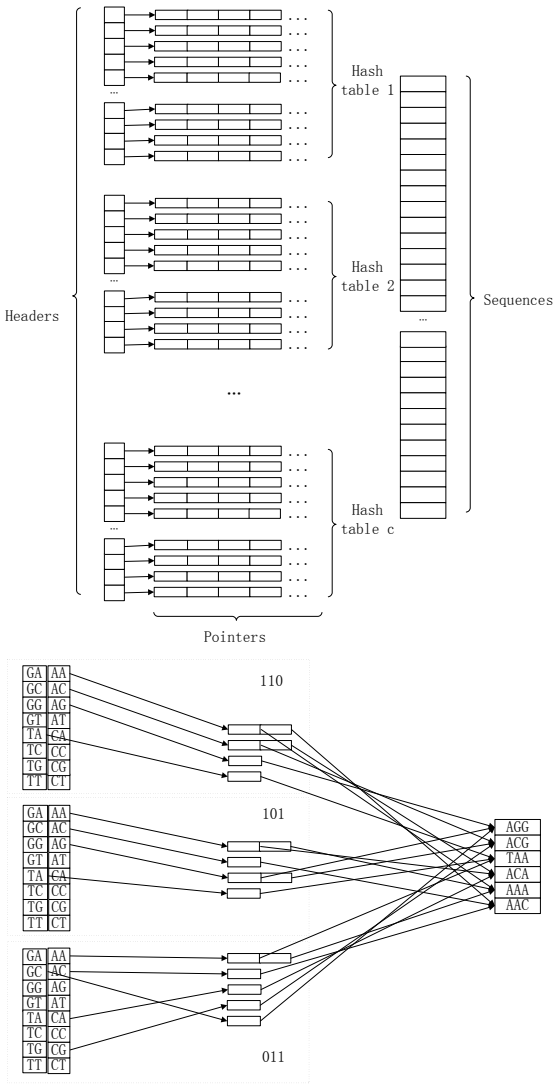


Figure A-1: Data structure of the SEED algorithm. The top figure illustrates the general data structure. On the left are the hash tables. Each hash table corresponds to one spaced seed and consists of buckets. Each bucket corresponds to a word of w bases and has two parts: (i) a header and (ii) an array of pointers the header points to. All the array pointers reference the sequence space on the right. The bopttom figure gives a specific example for 6 sequences with seeds $\{110, 101, 011\}$ (*i.e.*, $l = 3$, $k = 1$ and $w = 2$). There are three hash tables each corresponds to one spaced seed, and $4^2 = 16$ buckets in each hash table. Most of the buckets only have headers but not arrays of pointers, because the number of associated sequences is very small.

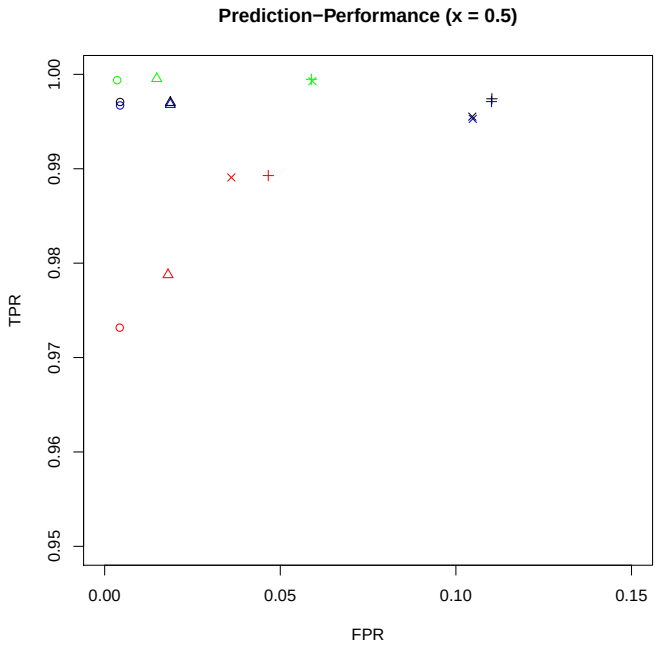


Figure A-2: Prediction-performance plot. The FPRs are plotted against the TPRs for the clustering methods: SEED (green), UCLUST with and without its *optimal* mode (blue), and SSAKE (red). The "true" clusters from the alignment-based method were used as reference to compute the data points for this plot. For each test method the results for the four ChIP-Seq samples are provided separately (SRR038848-SRR038851). The minimum similarity required to the true clusters was set to $x = 0.5$. Note, the results for two UCLUST modes overplot each other because they are almost identical.

Table A-1: Weights and numbers of block spaced seeds and memory usages of headers for various read lengths.

Read length	Weight	# block spaced seeds	Header memory usage
25	11	364	11.375 GB
26	12	84	10.5 GB
27	12	35	4.375 GB
28	12	20	2.5 GB
29	11	364	11.375 GB
30	12	10	1.25 GB
31	11	364	11.375 GB
32	12	20	2.5 GB
33	12	35	4.375 GB
34	12	84	10.5 GB
35	11	364	11.375 GB

Table A-2: The set of block spaced seeds used in the experiments. Each string of 1's and 0's is a block spaced seed.

11111111111100000000000000000
11111100000011111000000000000
11111100000000000011111000000
11111100000000000000000111111
00000011111111111000000000000
00000011111100000011111000000
00000011111100000000000111111
00000000000011111111111000000
00000000000011111000000111111
00000000000000000011111111111

Table A-3: Performance tests of SEED on simulated data. The default parameters are: 10^7 read sequences of length 40 bp, 1000 true clusters with 10,000 members each, ≤ 3 mismatches, ≤ 3 overhanging bases, QV1 = 0, QV2 = 558. Variations of these parameters are indicated in each subtable. The last two columns give the number of clusters containing at least 50% and 90% of their original members, respectively. The FPR for each result is provided in parentheses.

Test Variable	Memory in GB	Time	# ≥50% Clusters (FPR)	# ≥90% Clusters (FPR)
		(a) Varying numbers of sequences (N)		
N = 1×10^7	2.6	00:23:53	721 (0)	149 (0)
N = 4×10^7	4.6	01:36:55	713 (0)	155 (0)
N = 7×10^7	6.3	02:42:23	726 (0)	140 (0)
N = 1×10^8	8.0	03:53:08	725 (0)	159 (0)
		(b) Varying sequence lengths (L)		
L = 40	2.6	00:23:53	721 (0)	149 (0)
L = 60	2.6	00:38:17	723 (0)	133 (0)
L = 80	2.6	00:52:35	703 (0)	143 (0)
L = 100	2.6	01:05:48	707 (0)	165 (0)
		(c) Varying numbers of true clusters (TC)		
TC = 1	2.3	00:04:19	1 (0)	1 (0)
TC = 10	2.1	00:11:02	6 (0)	4 (0)
TC = 100	2.6	00:24:19	65 (0)	14 (0)
TC = 1000	2.6	00:23:43	721 (0)	149 (0)
		(d) Varying numbers of mismatches (MM)		
MM = 0	1.4	00:58:50	408 (0)	121 (0)
MM = 1	2.4	00:33:34	438 (0)	150 (0)
MM = 2	2.5	00:28:16	725 (0)	141 (0)
MM = 3	2.6	00:23:53	721 (0)	149 (0)
		(e) Varying numbers of overhanging residues (OH)		
OH = 0	2.5	00:05:28	1000 (0)	1000 (0)
OH = 1	2.6	00:07:02	1000 (0)	306 (0)
OH = 2	2.6	00:11:01	799 (0)	206 (0)
OH = 3	2.6	00:23:53	721 (0)	149 (0)
		(f) Varying threshold values for QV1 constraint		
QV1 = 0	3.0	00:41:47	721 (0)	149 (0)
QV1 = 23	3.0	00:32:37	721 (0)	149 (0)
QV1 = 70	3.0	00:16:23	933 (0)	149 (0)
QV1 = 93	3.0	00:10:48	1000 (8×10^{-6})	149 (2×10^{-5})
		(g) Varying threshold values for QV2 constraint.		
QV2 = 465	3.0	00:42:05	721 (0)	149 (0)
QV2 = 488	3.0	00:37:24	721 (0)	149 (0)
QV2 = 535	3.0	00:37:20	721 (0)	149 (0)
QV2 = 558	3.0	00:37:20	721 (0)	149 (0)

Appendix B

Supplementary Materials for Chapter 3

The Details of the CO Algorithm

Recall that the CO Algorithm solves the MMPC problem. The standard way to solve the Minimum Path Cover (MPC) problem is to convert the DAG G to a bipartite graph G' and solve the corresponding Maximum Matching (MM) problem for G'. Therefore, to solve the MMPC problem, we could also convert the weighted DAG G to a weighted bipartite graph G', and solve the correspoding Minimum weight Maximum Matching (MMM) problem (Trapnell *et al.*, 2010). Due to the lack of a suitable software library for solving the MMM problem available to us at the time of this study, we further convert the weighted bipartite graph G' to a weighted complete graph G'' and then solve the corresponding Maximum Perfect Matching (MPM) problem, which can be efficiently solved by using several public

software libraries (*e.g.* Dezso *et al.*, 2011). A pseudocode for the CO Algorithm is given below.

Algorithm 4 Combinatorial Optimization: CO(G)

```
convert G to bipartitie graph G'
for each node pair v and v' ∈ G' do
    if there exists an edge e(v, v') then
        let the weight w(e(v, v')) = −w(e(v, v'))
    else
        assign an edge e(v, v') with weight −MAX to G
    end if
end for
{G' is then converted to a complete graph G''}
MPM(G'')
for each edge e do
    if w(e) = −MAX then
        delete e
    else
        w(e) = −w(e)
    end if
end for
{G'' is converted back to G'}
convert G' back to G and return the resultant MMPC P
```

The following lemma can be proven by a straightforward proof-by-contradiction argument and guarantees the correctness of the CO Algorithm.

Lemma 1. *An MPM on G'' in the CO Algorithm corresponds to an MMM on G'.*

Table B-1: Population sizes used for mRNA read sampling. The proportions of transcripts are listed falling into variable ranges of expression levels. The latter are given in reads per kb of exon model per million reads (RPKM).

RPKM	0	0-0.1	0.1-1	1-10	10-100	>100
N Transcripts	50%	16%	10%	14%	8%	2%

Table B-2: Assemblies of simulated data with variable contig lengths. Performance results for different contig lengths are shown for Velvet/Oases, Velvet/Oases with BRANCH post-processing (referred to as Velvet/Oases+BRANCH), Trinity, and Trinity with BRANCH post-processing (referred to as Trinity+BRANCH). The invariable parameter settings include: 1% contig sequencing errors, 80% contig coverage, 50 million paired-end RNA reads, and 1% RNA read base call errors. The corresponding sensitivity plot is given in Figure 3.3a.

Contig length	Sensitivity	Precision	N Transfrags	N Complete transcripts	N Complete genes	N Covered transcripts	N Exons	N Extensions
				Velvet/Oases				
-	4,235	17.7%	85,082	1,736	1,585	1,994	50,350	-
				Velvet/Oases+BRANCH				
1K	4,403	24.6%	95,893	1,751	1,598	2,350	55,537	1,758
10K	4,769	22.1%	97,709	1,821	1,647	2,369	57,589	3,627
50K	4,947	21.4%	97,152	1,871	1,698	2,483	58,063	4,146
100K	4,620	19.4%	90,821	1,812	1,650	2,225	54,230	2,034
				Trinity				
-	4,817	39.1%	18,241	2,798	2,538	3,022	53,659	-
				Trinity+BRANCH				
1K	4,929	54.8%	28,536	2,808	2,547	3,143	57,755	487
10K	5,287	46.5%	28,794	2,805	2,531	3,186	59,490	1,403
50K	5,340	44.3%	28,682	2,840	2,564	3,191	59,819	1656
100K	5,077	42.0%	23,213	2,827	2,558	3,102	56,863	826

Table B-3: Assemblies of simulated data with variable contig base error rates. The invariable parameter settings include: 10 kbp contig length, 80% contig coverage, 50 million paired-end RNA reads, and 1% RNA read base call errors. The corresponding sensitivity plot is given in Figure 3.3b.

Error rate	Sensitivity	Precision	N Transfrag	N Complete transcripts	N Complete genes	N Covered transcripts	N Exons	N Extensions
				Velvet/Oases				
-	4,235	17.7%	85,082	1,736	1,585	1,994	50,350	-
				Velvet/Oases+BRANCH				
0%	4,827	22.0%	96,733	1,869	1,695	2,410	58,038	3,982
1%	4,769	22.1%	97,709	1,821	1,647	2,369	57,589	3,627
2%	4,663	22.1%	98,311	1,778	1,620	2,337	57,292	3,257
3%	4,625	21.9%	98,526	1,766	1,602	2,283	56,498	3,033
				Trinity				
-	4,817	39.1%	18,241	2,798	2,538	3,022	53,659	-
				Trinity+BRANCH				
0%	5,317	47.6%	28,024	2,838	2,562	3,209	59,577	1,441
1%	5,287	46.5%	28,794	2,805	2,531	3,186	59,490	1,403
2%	5,219	45.9%	29,450	2,814	2,539	3,170	59,090	1,301
3%	5,215	45.2%	29,603	2,793	2,524	3,157	58,743	1,257

Table B-4: Assemblies of simulated data with variable genome coverage by contigs. See legend of Table S-2 and S-3 for details. The corresponding sensitivity plot is given in Figure 3.3c.

Contig coverages	Sensitivity	Precision	N Transfrag	N Complete transcripts	N Complete genes	N Covered transcripts	N Exons	N Extensions
				Velvet/Oases				
-	4,235	17.7%	85,082	1,736	1,585	1,994	50,350	-
				Velvet/Oases+BRANCH				
40%	4,597	20.1%	91,127	1,807	1,640	2,251	54,352	1,930
60%	4,753	21.1%	93,905	1,839	1,667	2,355	56,013	2,790
80%	4,769	22.1%	97,709	1,821	1,647	2,369	57,589	3,627
100%	5,077	23.0%	100,099	1,892	1,706	2,606	59,682	4,651
				Trinity				
-	4,817	39.1%	18,241	2,798	2,538	3,022	53,659	-
				Trinity+BRANCH				
40%	5,092	43.6%	23,631	2,804	2,536	3,110	56,968	812
60%	5,207	45.2%	26,091	2,805	2,536	3,157	58,063	1,125
80%	5,287	46.5%	28,794	2,805	2,531	3,186	59,490	1,403
100%	5,445	47.7%	31,478	2,816	2,538	3,254	60,889	1,824

Table B-5: Assemblies of simulated data with variable numbers of RNA reads. See legend of Table S-2 and S-3 for details. The corresponding sensitivity plot is given in Figure 3.3d.

N RNA reads	Sensitivity	Precision	N Transfrags	N Complete transcripts	N Complete genes	N Covered transcripts	N Exons	N Extensions
				Velvet/Oases				
10M	3,724	15.6%	54,951	1,444	1,296	1,539	40,185	-
30M	4,041	18.0%	76,172	1,708	1,552	1,908	47,466	-
50M	4,235	17.7%	85,082	1,736	1,585	1,994	50,350	-
70M	4,461	18.9%	88,993	1,863	1,697	2,147	51,432	-
				Velvet/Oases+BRANCH				
10M	3,978	20.7%	61,824	1,443	1,292	1,713	44,645	1,673
30M	4,464	22.5%	87,066	1,743	1,580	2,226	53,639	3,153
50M	4,769	22.1%	97,709	1,821	1,647	2,369	57,589	3,627
100M	4,948	23.1%	103,052	1,920	1,742	2,547	59,479	3,897
				Trinity				
10M	3,702	42.2%	13,461	1,839	1,671	1,987	40,992	-
30M	4,459	42.9%	16,486	2,576	2,326	2,734	49,807	-
50M	4,817	39.1%	18,241	2,798	2,538	3,022	53,659	-
70M	5,035	39.4%	19,229	3,066	2,782	3,284	55,067	-
				Trinity+BRANCH				
10M	4,031	49.4%	19,459	1,810	1,643	2,072	45,301	1,010
30M	4,833	49.4%	25,468	2,584	2,329	2,868	55,066	1,419
50M	5,287	46.5%	28,794	2,805	2,531	3,186	59,490	1,403
70M	5,478	46.1%	31,309	3,102	2,799	3,449	61,769	1,457

Table B-6: Assemblies of simulated data with variable RNA read base call error rates. See legend of Table S-2 and S-3 for details. The corresponding sensitivity plot is given in Figure 3.3e.

Error rate	Sensitivity	Precision	N Transfrags	N Complete transcripts	N Complete genes	N Covered transcripts	N Exons	N Extensions
				Velvet/Oases				
0%	4,977	13.1%	66,796	2,545	2,297	2,651	51,276	-
1%	4,235	17.7%	85,082	1,736	1,585	1,994	50,350	-
2%	4,275	21.6%	99,080	1,453	1,326	1,771	49,588	-
3%	4,281	24.2%	111,420	1,315	1,177	1,662	47,565	-
				Velvet/Oases+BRANCH				
0%	5,187	16.1%	71,558	2,502	2,247	2,743	54,647	1,227
1%	4,769	22.1%	97,709	1,821	1,647	2,369	57,589	3,627
2%	4,771	24.7%	113,578	1,532	1,396	2,187	56,778	4,707
3%	4,735	26.3%	127,631	1,399	1,251	2,167	54,662	5,394
				Trinity				
0%	5,019	38.8%	18,769	2,972	2,671	3,169	54,889	-
1%	4,817	39.1%	18,241	2,798	2,538	3,022	53,659	-
2%	4,461	40.9%	17,784	2,591	2,371	2,794	50,511	-
3%	4,246	42.5%	17,045	2,502	2,282	2,703	47,850	-
				Trinity+BRANCH				
0%	5,357	42.7%	23,051	2,916	2,606	3,268	59,118	1,206
1%	5,287	46.5%	28,794	2,805	2,531	3,186	59,490	1,403
2%	4,987	46.4%	29,014	2,627	2,398	3,015	57,210	1,722
3%	4,739	46.2%	28,576	2,533	2,306	2,885	54,539	1,926