# KOLLECTOR: TRANSCRIPT-INFORMED TARGETED *DE NOVO* ASSEMBLY OF GENE LOCI

by

Muhammet Erdi Kucuk

B.S., Bilkent University, 2013

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Bioinformatics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

April 2017

## Abstract

The information stored in nucleotide sequences is of critical importance for modern biological and medical research. However, in spite of considerable advancements in sequencing and computing technologies, *de novo* assembly of whole eukaryotic genomes is still a time-consuming task that requires a significant amount of computational resources and expertise, and remains beyond the reach of many researchers. One solution to this problem is restricting the assembly to a portion of the genome, which is typically a small region of interest. Genes are the most obvious choice for this kind of targeted assembly approach, as they contain the most relevant biological information, which can be acted upon downstream. Here we present Kollector, a targeted assembly pipeline that assembles genic regions using the information from the transcript sequences. Kollector not just enables researchers to take advantage of the rapidly expanding transcriptome data, but is also scalable to large eukaryotic genomes. These features make Kollector a valuable addition to the current crop of targeted assembly tools, a fact we demonstrate by comparing Kollector to the state-of-the-art. Furthermore, we show that by localizing the assembly problem, Kollector can recover sequences that cannot be reconstructed by a whole genome *de novo* assembly approach. Finally, we also demonstrate several use cases for Kollector, ranging from comparative genomics to viral strain detection.

## Preface

The work presented in this thesis is conducted in Canada's Michael Smith Genome Sciences Research Centre under the supervision of Dr. Inanc Birol. Dr. Birol conceived the project idea. M. Erdi Kucuk was responsible for designing, developing and implementing the Kollector pipeline, and also ran all the experiments that are described in Chapter 3. Justin Chu designed, developed and implemented the progressive Bloom filters.

A paper describing Kollector pipeline has been published, and includes all the results presented in Chapter 3, except section 3.3.4:

Erdi Kucuk, Justin Chu, Benjamin P Vandervalk, S Austin Hammond, René L Warren, Inanc Birol; Kollector: transcript-informed, targeted *de novo* assembly of gene loci. *Bioinformatics* 2017 btx078. doi: 10.1093/bioinformatics/btx078

## Table of Contents

## List of Tables

## List of Figures

## List of Abbreviations

| | |
|---|---|
| ABySS | Assembly by Short Sequences |
| aTRAM | Automated Target Restricted Assembly Method |
| BART | Bullfrog Annotation Reference Transcriptome |
| BBT | BioBloom Tools |
| BF | Bloom Filter |
| BLAST | Basic Local Alignment Tool |
| BWA | Burrows-Wheeler Align |
| cDNA | Complementary DNA |
| CEG | Core Eukaryotic Gene |
| CEGMA | Core Eukaryotic Genes Mapping Approach |
| Contig | Contiguous (Sequence) |
| DBG | De Bruijn Graph |
| ddNTP | Dideoxynucleotide |
| DNA | Deoxyribonucleic Acid |
| Gbp | Gigabase Pairs |
| GMAP | Genomic Mapping and Alignment Program |
| GNU | GNU's not Unix! |
| GRAbB | Genomic Region Assembly by Baiting |
| HGP | Human Genome Project |
| HMM | Hidden Markov Model |
| HPC | High-Performance Computing |
| HPV | Human Papilloma Virus |
| ICGC | International Cancer Genome Consortium |
| Indel | Insertions and Deletions |
| Kbp | Kilobase Pairs |
| Mbp | Megabase Pairs |
| mRNA | Messenger RNA |
| MUSCLE | Multiple Sequence Comparison by Log-Expectation |
| NCBI | National Center for Biotechnology Information |
| NGS | Next-generation Sequencing |
| OLC | Overlap-Layout-Consensus |
| pBF | Progressive Bloom Filter |
| PE | Paired-end |
| PET | Paired-end Tagged |
| QC | Quality Control |
| Refseq | Reference Sequence Database |
| RNA | Ribonucleic Acid |

| | |
|---|---|
| RNA-Seq | RNA Sequencing |
| SAM | Sequence Alignment/Map |
| SNV | Single Nucleotide Variant |
| SOAPdenovo | Short Oligonucleotide Analysis Package *de novo* |
| TASR | Targeted Assembly of Short Reads |
| TCGA | The Cancer Genome Atlas |
| Unitig | Uniquely Assemble-able Contigs |
| USD | United States Dollars |
| WGSS | Whole Genome Shotgun Sequencing |

# Chapter 1: Introduction

In this chapter we begin by providing a historical overview of DNA sequencing technologies and the sequence assembly methods in **Section 1.1** and **Section 1.2**. Despite researchers' best efforts to improve, the assembly problem is not yet solved, and the assembly of large eukaryotic genomes remains a computationally challenging task. These challenges sparked the development of targeted assembly approaches to perform local assembly of sequences of interest. We review the current crop of targeted assembly tools in **Section 1.3**. We then state our hypothesis as follows: utility of the targeted assembly approach can be greatly improved by taking advantage of different types of data and increasing the scalability to large eukaryotic genomes. Our main contribution, as summarized in **Section 1.4**, is Kollector, a targeted assembly pipeline that can utilize transcript sequences to assemble corresponding genic loci from whole genome shotgun sequencing data.

## 1.1 DNA sequencing

Knowledge of DNA sequences is indispensable in modern biological and medical research, as nucleotides carry the hereditary and biochemical information in all terrestrial organisms. However, the researchers' ability to "sequence" or determine the precise order of nucleotides in a DNA molecule remained severely limited until 1977. In that year Sanger and colleagues achieved a major breakthrough with the development of chain-termination method (Sanger, et al., 1977). This technique relies on mixing radiolabelled dideoxynucleotides (ddNTPs) at a fraction of standard nucleotides into a DNA synthesis reaction. When incorporated into newly synthesized DNA, ddNTPs halt further extension, thus generating fragments of every possible length. By performing four parallel reactions with each ddNTP base and analyzing the length of the fragments with autoradiography, nucleotide sequence of the original template can be inferred.

The accuracy and robustness of dideoxy chain-termination method, which came to be called simply Sanger sequencing, led to its widespread adaptation as the most common method and the gold standard for DNA sequencing in the next 40 years (Heather and Chain, 2016).

Sanger method can be used to sequence DNA fragments or "reads" up to 1-kilobase pairs in length. In order to analyze longer sequences, researchers developed the 'shotgun sequencing' approach where the DNA molecule is fractured into overlapping fragments that are amplified and sequenced separately, and then reconstructed into the longer target sequence with computational methods (see **Section 1.2**). Shotgun sequencing came to be used in Human Genome Project (HGP), and is still the standard approach for whole genome sequencing efforts. However, Sanger sequencing proved to be a prohibitively expensive method for such large-scale projects, as exemplified by the cost of HGP around USD $3 billion (Heather and Chain, 2016). As a result, the turn of the 21$^{st}$ century saw the development of higher throughput and thereby lower cost DNA sequencing technologies as alternatives.

### 1.1.1 Next-generation sequencing

Next-generation sequencing (NGS) refers to the sequencing technologies that have become commercially available starting over a decade ago to replace Sanger sequencing for large-scale applications. Compared to Sanger sequencing NGS methods typically have shorter read lengths, but higher depth of coverage, which is defined as the number of times a nucleotide is read during sequencing (Goodwin, et al., 2016). This is achieved by relying on a massively parallel sequencing strategy, which involves random fragmentation and clonal amplification of DNA.

An example of this would be the Solexa method of sequencing, a technology later acquired by the current market leader Illumina (Chaisson, et al., 2015). In the Solexa method, template DNA is first sheared into small (<1 kb) fragments, and small oligonucleotide sequences

called adapters are attached to both ends. These fragments, bracketed with adapters, are then passed over a flow cell with a lawn of oligonucleotides complimentary to the adapter sequence. Once bound, DNA fragments are amplified on the flow cell, generating clusters of clonal DNA. Sequencing itself is achieved during DNA replication, in which "reversible-terminator" nucleotides are used. These nucleotides contain a terminator domain, which halts the extension of the DNA strand that it is incorporated. The terminator domain also contains a fluorescent label, which can be detected with a camera. Since only one fluorescent color is used, each of the four bases must be added in separate cycles. In each cycle, nucleotides and DNA polymerases are washed off from the flow cell and incorporation of the nucleotide is determined by checking the fluorescent signal. After four cycles, the terminator domain is enzymatically cleaved off from the incorporated nucleotides, so sequencing can continue with the subsequent position in the DNA template for the next four cycles (Bentley, et al., 2008).

Solexa/Illumina platform produces short (<300 base pairs) but highly accurate (1% error rate) reads. In order to compensate for the short read length, the Illumina platform can generate paired-end (PE) data, where matched reads contain sequence from opposite ends of the same DNA fragment. In paired-end sequencing, after sequencing one end of the bound DNA in the flow cell, a DNA extension step is performed and already-sequenced strand is removed. Since orientation of the template DNA is reversed, a second read can be obtained from the opposite end. As all input DNA fragments are of a known approximate length, the distance between the first and the second read can also be estimated. This extra information aids in the downstream processing of the PE reads, such as sequence assembly or mapping to a reference genome for structural variant detection (Voelkerding, et al., 2009).

In terms of application, NGS is mainly employed in two types of projects: *de novo* sequencing and re-sequencing (Goodwin, et al., 2016). In *de novo* sequencing projects, genome of an organism is sequenced for the first time and typically the goal is to reconstruct the whole genome using sequence assembly. This reconstructed genome is then can be used as a representative for the species in question, and is called a reference genome. In contrast, re-sequencing projects involve partial or whole sequencing of a genome from a species where a reference genome is already available. Re-sequencing is usually performed to characterize and study genomic variations in certain populations and individuals, such as cancer patients. Both *de novo* sequencing and re-sequencing rely on computational methods for downstream analysis and processing of the NGS data. Chief among them is the sequence assembly, which is described in the next section.

### 1.2 The assembly problem

Sequencing methods that were introduced in the previous section can typically yield reads with lengths ranging from 20 to 30000 base pairs, depending on the technology used. On the other hand, genomic features that interest researchers, such as genes, can typically range from a couple hundred to several millions of base pairs in length. Thus, sequence assembly methods have been developed to reconstruct longer sequences from the reads computationally, or *in silico*.

To date, two main strategies have been employed in sequence assembly tools: the mapping approach, in which an existing reference sequence is used as a backbone to align and orient reads, and the *de novo* assembly approach, in which the full-length sequence is reconstructed solely *in silico* from the input reads (Chaisson, et al., 2015).

### 1.2.1 *De novo* assembly

*De novo* assembly is a necessity for most research projects due to absence of high-quality reference genomes for many non-model organisms. Even in cases where a reference genome is available, using a mapping approach results in an inherent bias towards the reference sequence used. Some degree of ambiguity also occurs due to multi-mapping of the reads. Therefore, for some applications like structural variant calling, a de *novo* assembly approach might be preferred for better characterization (Alkan, et al., 2011).

Two main computational strategies used for *de novo* assembly are overlap-layout-consensus (OLC) and de Bruijn graphs (Nagarajan and Pop, 2013). In the OLC approach, reads that overlap "sufficiently" (usually beyond a certain length) are organized into a graph, in which each read is a node, and overlapping reads are connected by edges. This graph is then traversed by the algorithm to reconstruct global sequences. First popularized by Gene Myers and his Celera assembler (Myers, et al., 2000), the viability of OLC became strained with the advent of high-throughput short-read sequencing of NGS, since time-and-memory requirements of OLC algorithms scales quadratically with the number of reads. This led to the development of de Bruijn graph (DBG) assemblers, starting with Euler Pevzner, et al. (2001) and continuing with Velvet (Zerbino and Birney, 2008), ABySS (Simpson, et al., 2009) and SOAPdenovo (Li, et al., 2010). These assemblers extract substrings of length k, called k-mers, from the input reads and neighboring information for each k-mer is recorded. Just like OLC, a graph is built in which k-mers correspond to nodes while edges represent exact k-1 base pair overlaps between k-mers. The main advantage DBG approach is that it avoids the computationally intensive multiple sequence alignment step that is necessary for the OLC (Li, et al., 2012). For this reason, DBG

remains the dominant paradigm in modern assembler design, especially when dealing with short, highly accurate reads with high coverage. (Chaisson, et al., 2015).

### 1.2.2 Evaluating *de novo* assemblies

In the last decade, combination of NGS data and DBG assemblers has been used to produce *de novo* assemblies of genomes ranging in size from bacterial to that of giant panda. As more and more researchers began to rely on information provided by the *de novo* assembly approach, the critical assessment and evaluation of *de novo* assemblies became more crucial. Unsurprisingly, due to the limitations of the NGS technology and relative youth of the DBG assemblers, these assemblies have certain shortcomings compared to the reference genomes produced by traditional Sanger sequencing method. Chief among these shortcomings are fragmentation and assembly mistakes.

Fragmentation of *de novo* assemblies often occurs due to presence of low coverage regions, read errors or the inability of assembler to resolve repeat sequences (Baker, 2012). It is not unusual for a *de novo* assembly of a human-size genome to include millions of contigs. Therefore, an important goal of genome assembly process is increasing contiguity of the genome assembly, which is often measured by the N50 metric. N50 is defined as the length of the shortest contig in the set that contains fewest (longest) contigs that represent at least 50% of the assembly sequence. If the genome size is known, an alternative metric, NG50, can be defined as the length of the shortest contig in the set of longest contigs that represent at least 50% of the genome (Nagarajan and Pop, 2013).

Limitations of NGS data (coverage, read errors) and DBG assemblers (resolving repeat regions) also cause mistakes in *de novo* assemblies, which are called mis-assemblies (Alkan, et al., 2011). Reported deficiencies of *de novo* assemblies typically include missing sequences,

nucleotide changes, contamination, order errors, orientation errors and systemic collapse of segmental duplications (Birney, 2011). Oftentimes, these assembly errors are only detectable if a high-quality reference genome is already available for comparison. For *de novo* sequencing projects, researchers have to use different methods for evaluating completeness and correctness of an assembly. These often make use of comparative genomics and independently derived data, such as using genomes of closely related organisms and transcriptomes. For instance, Core Eukaryotic Genes Mapping Approach (CEGMA) method takes advantage of the conserved sequences. CEGMA uses a statistical model built from 248 most conserved eukaryotic genes and detects these genes in a given set of sequences (Parra, et al., 2007). The number of Core Eukaryotic Genes (CEGs) found in an assembly is frequently reported as a metric of completeness. This is an important metric because improving genic reconstruction of draft genomes is a valuable target for computational biologists.

### 1.3 Targeted assembly

*De novo* assembly methods provided valuable insights for biological research, especially for the study of non-model organisms and characterization of large-scale genomic variants (Nagarajan and Pop, 2013). However, the *de novo* assembly process still involves significant computational challenges. Production of high quality reference genome sequences for non-model organisms remains a challenging endeavor, especially for large (>1Gbp) genomes. For such targets, *de novo* whole-genome assembly typically requires billions of sequencing reads from several different types of DNA libraries. Processing these large volumes of data, and using them to assemble a genome usually necessitates access to a high-performance computing environment and significant expertise with specialized software, which may be beyond the reach of many researchers (Nagarajan and Pop, 2013).

An attractive alternative to the production of a reference genome is the targeted assembly of gene sequences. Most of the sequence features that contain the most valuable biological information and therefore interest researchers, such as genomic variants and coding sequences, are located in the genic regions. Furthermore, most researchers in the life sciences focus their studies on a limited number of genes or gene families, so they can benefit from lower computational cost of a targeted approach. This is especially true for researchers working on non-model organisms that lack a reference genome. Obtaining high-quality gene sequences from these species is the first and most crucial step for many applications, such as comparative genomics and experimental design for further studies.

In addition to the benefit of the low computational cost, the targeted assembly approach also reduces the complexity of the assembly graph, since only a subset of reads is used. Ideally, this can lead to more contiguous assemblies with fewer mistakes compared a whole genome assembly approach. For this reason, targeted assembly is frequently used for precise characterization of structural variants, such as gene fusion events (Warren, et al., 2012).

The majority of targeted assembly tools are designed for whole genome shotgun sequencing (WGSS) data and follow the same basic pattern of sequence categorization and assembly of categorized reads. In the first step, a user-provided sequence template is utilized to categorize reads in the WGSS dataset. The reads belonging to the target region are then used for the assembly, which can be *de novo* assembled or make use of the template sequence for mapping. Since the target region is often limited to a single gene, the computational cost of the actual assembly is not significant compared to the sequence categorization, which remains the bottleneck for the targeted assemblers described in the next section.

### 1.3.1 Existing tools

The first solution for the reconstruction of specific targets was the k-mer based, alignment-free, targeted *de novo* assembly software **T**argeted **A**ssembly of **S**hort **R**eads, or TASR (Warren and Holt, 2011). TASR requires the entire target sequence as input, from which k-mers are extracted and stored in a data structure called a hash table. Hash tables map input keys (in this case k-mers) to values (in this case, how many times they appeared) by using a hash function to compute an index into an array of values. Once the hash table is built, TASR k-merizes the genomic reads and uses the same hash function to look up any shared k-mers with the input sequence, thus localizing the assembly. A second hash table is then created from collected reads, which includes the identity and coverage of every base. This second table is used to derive a consensus sequence for the target region.

This method was followed by Mapsembler (Peterlongo and Chikhi, 2012), which also uses a hash table based mapping method, and presents a more memory-efficient and faster alternative. Just like TASR, Mapsembler's first step is mapping reads to the input sequence (starter). However, instead of deriving a single consensus Mapsembler does an error correction of collected reads and generates sub-starter sequences, which may differ from the initial starter sequences with single nucleotide variants (SNVs) and insertions and deletions (indels). Genomic reads are then re-mapped into sub-starters, which iteratively extend flanking regions of these sequences.

These pioneering targeted assembly technologies were originally designed to reconstruct specific transcript variants, fusion transcripts, and genes from whole genome shotgun sequencing data, but now have found applications in human health research (Brown, et al., 2014; Warren, et al., 2012). Unfortunately, these applications are mostly confined to re-sequencing projects, as

these initial targeted assembly tools struggle when an incomplete or divergent sequence is used to localize reads for assembly. Furthermore, for eukaryotic genomes, these tools do not typically scale well when given a large (>1000) number of targets.

Recently published **a**utomated **T**arget **R**estricted **A**ssembly Method or aTRAM (Allen, et al., 2015), is an alignment-based pipeline that can deal with incomplete input sequences, to some extent. aTRAM uses **B**asic **L**ocal **A**lignment **T**ool, or BLAST (Altschul, 1990) for aligning reads to a user-provided nucleotide or protein sequence. Aligned reads are then assembled with a *de novo* assembler, by default Velvet (Zerbino and Birney, 2008) for genomic sequences and Trinity (Grabherr, et al., 2011) for transcript sequences. These newly assembled sequences are then used as bait for the next iteration of BLAST alignments, ideally extending the resulting assemblies by the fragment length in each iteration. Users can specify higher number of iterations to further extend final assemblies, but this also increases the time cost. In order to alleviate the time cost that is associated with high number of iterations, aTRAM allows for random subsampling of the read set before the alignment, which reduces the time complexity for that step. However, subsampling also decreases the coverage of the recruited reads, which might not be desirable if the target region has low coverage to begin with.

The main advantage of aTRAM is that it can start with a partial or divergent template sequence and extend it in every iteration. However, the need for an alignment step in each iteration exerts a significant computational cost. This severely limits the scalability of aTRAM as WGSS datasets of complex eukaryotic organisms typically include billions of reads.

Another recently published too, **G**enomic **R**egion **A**ssembly by **B**aiting (GRAbB) also utilizes iterative recruitment of the reads from target regions, thus allowing it to fill in the gaps in the case of an incomplete template sequence (Brankovics, et al., 2016). GRAbB uses a special

aligner called mirabait (Chevreux, 1999) to find reads that share a 31-mer (subsequence of 31 base pairs length) in common with the target. The recruited reads are assembled with a *de novo* assembler of the user's choice, the default option being Edena (Hernandez, et al., 2008). Assembled sequences are then used as bait for the next iteration. Users have several options for finishing criteria. Iterations can be stopped when the assembly N50 or longest contig legth reach a user-defined threshold, or original bait is completely mapped to the assembly with Exonerate (Slater and Birney, 2005). GRAbB also supports multiple targets as bait. In this mode, a general recruitment step is performed first using all the input sequences. This is followed by a specialized recruitment step, where each target is used as a separate bait to recruit from the pooled reads from the previous general recruitment step. Thus, search space for individual targets is reduced. Despite this heuristic, scalability of GRAbB to complex eukaryotic genomes is very limited. This is especially true when transcript sequences are used as bait for genic regions, since assembling long introns requires multiple iterations.

### 1.4 Current work

One resource that is currently under-utilized for targeted gene assembly is transcriptomic data. These datasets are typically obtained with an approach called RNA sequencing or RNA-Seq, where NGS methods are used to sequence cDNA libraries (Wang, et al., 2009). Last decade also saw the development of several specialized assemblers for RNA-seq datasets, foremost among them are Trinity (Grabherr, et al., 2011), Trans-ABySS (Robertson, et al., 2010) and IDBA-tran (Peng, et al., 2013). As a result, expansive transcriptomic information from many organisms is currently available to researchers (Moreton, et al., 2015), Even in species with scant transcriptomic sequence information, there are likely existing sequences that could be used to aid *de novo* assembly, such as homologous gene sequences from a related organism. Utilization of

these data can help to localize the assembly problem, and can ensure that the target gene sequences are fully reconstructed. A desirable consequence of this localization is a reduction in complexity and computational cost relative to that of a whole genome assembly.

We also believe that the targeted gene assembly approach can be utilized to improve the reconstruction quality of genic sequences in draft genomes. Compared to the whole-genome assembly process, targeted assembly benefits from reduced complexity and can take advantage of the additional information in the user-provided templates. In practice, however, use of targeted assembly approaches in *de novo* genome assembly projects remains limited. This is because the computational cost of identifying reads related to the target sequences is challenging for genome-scale applications in eukaryotic organisms, as it often scales linearly with the increasing number of targets. Furthermore, as explained above, existing tools are often unable take full advantage of different kinds of datasets such as transcriptomes, which limits their usefulness in improving genic reconstruction. With these considerations in mind, we have developed Kollector, an alignment-free, targeted assembly pipeline that can use whole transcriptome assemblies to filter whole genome shotgun sequencing reads, thereby localizing the *de novo* assembly of corresponding genic loci.

The pipeline collects genomic reads related to target loci using BioBloom Tools (BBT) (Chu, et al., 2014), a Bloom filter (Bloom, 1970) based sequence classification tool. Just like hash tables, Bloom filters use hash functions to query whether an element is a member of a set. Unlike hash tables, the set is represented by a bit array with hash-indexed memory locations, significantly reducing the memory requirement. For the assembly step, Kollector uses ABySS (Simpson, et al., 2009), a de Bruijn graph short read *de novo* assembler (Pevzner, et al., 2001).

In this work we characterize the Kollector pipeline, compare it to the state-of-the-art and showcase several possible applications. We show that that Kollector can accurately assemble genic loci in *Homo sapiens* (Human) and model organism *Caenorhabtidis elegans* (roundworm). When using transcript input, Kollector outperforms other targeted assemblers in significantly less runtime. We demonstrate that Kollector can reasonably scale up to genome-wide applications in *Picea glauca,* which has an estimated genome size around 20 Gb (Warren, et al., 2015). We also successfully employed Kollector for improving genic reconstruction of the *Rana catesbeiana* draft genome. Other applications include medical and comparative genomics.

We believe Kollector represents an improvement upon the state-of-the-art with its innovative use of Bloom filters. Most importantly, it opens up a myriad of new applications for the targeted assembly approach. As the sequencing continues to grow exponentially, we expect the interest for tools like Kollector to increase as well.

## Chapter 2: The Method

In this chapter, we provide a detailed description of the Kollector pipeline, and the tools and the technologies it uses. Kollector inputs a set of target transcripts and whole genome shotgun sequencing data, and reconstructs the corresponding genic space in four basic stages: (1) read tagging with a progressive Bloom filter, where genomic reads coming from the targeted regions are detected and stored in a Bloom filter (**Section 1.1**); (2) read recruitment with the Bloom filter from the first stage, where genomic reads are scanned for sequence similarity; (3) *de novo* assembly of recruited reads (**Section 1.2**); and (4) alignment of the transcripts to the assembly for

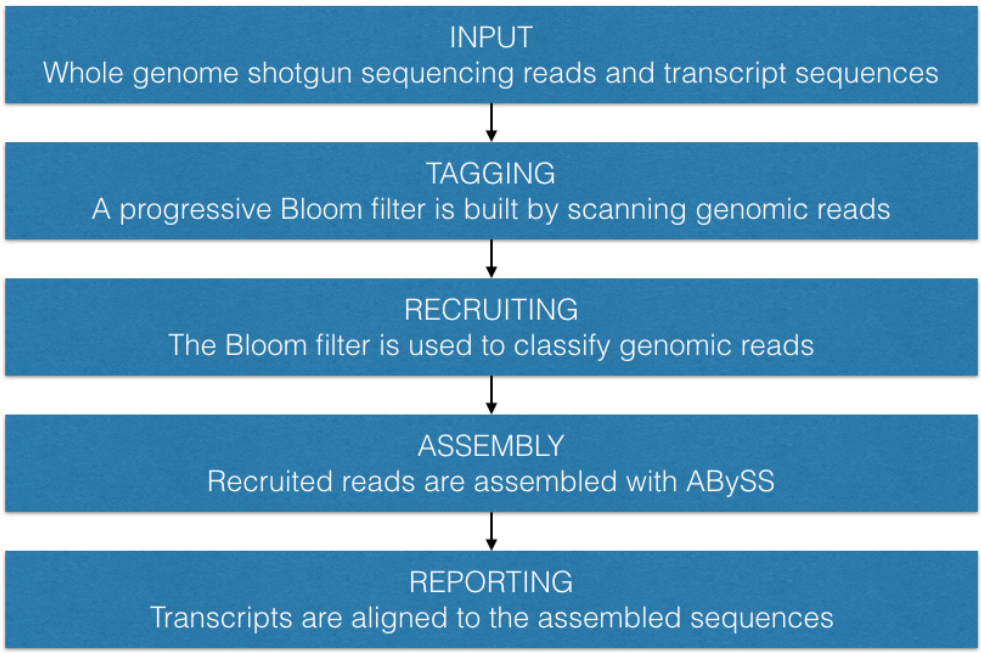



**Figure 2.1 Overview of the Kollector Pipeline**

evaluation and post-processing of the results to report assemblies containing the targeted regions (**Section 1.3**). The performance and efficiency can be improved by filtering repeat sequences with a dedicated Bloom filter during the tagging stage, which is described in **Section 1.4.** Finally, implementation and availability of Kollector software is discussed in **Section 1.5**.

## 2.1 Read tagging with progressive Bloom filters

The first stage of Kollector pipeline is a sequence categorization step performed by BioBloom Tools (BBT) (Chu, et al., 2014). In order to incorporate intronic sequences, a novel implementation of Bloom filter data structure was developed within BBT. We called this implementation the progressive Bloom filter. Using this data structure, BBT detects ("tags") reads that contain the target genic sequences, based on their sequence similarity to the input transcripts. The following is a description of the Bloom filter data structure, BBT and progressive Bloom filter implementation

### 2.1.1 Bloom filters

Bloom filters are memory-efficient probabilistic data structures that are used to test whether an element is part of a set (Bloom, 1970). A query can return false positive matches, but false negatives are not possible. Bloom filters are typically applied to large datasets that would require significant amount of space in the memory to store. In order to reduce this memory requirement, Bloom filters rely on a special type of mathematical functions that are called hash functions. Hash functions deterministically map data of an arbitrary size to data of a fixed size. For example, a hash function can take phonebook entries as input and return integers that range from 0 to 100 for each entry. The returned values are called hash values, or simply hashes. Since hash values have a fixed range, the domain of a hash function (set of possible inputs, or keys) is typically much larger than its range (number of possible hash values). Therefore, several

different keys will map to the same hash value, a situation that is termed collision. Ideally, a good hash function will map its expected inputs as evenly as possible over its output range, thus reducing the number of collisions as much as possible.

Hash functions are essential ingredients of Bloom filters, as they are used to insert or query new elements. Bloom filters store the input set in an array of *m* bits, initially all set to 0. An element is inserted into the Bloom filter by inputting it through *n* different hash functions. Each hash function returns a numerical value corresponding to an array position. The element is inserted to the Bloom filter by setting those *n* array positions to 1. To query the presence of an element in the Bloom filter, the same *n* hash functions are used to get *n* array positions for the queried element. Since hash functions are deterministic, inputting the same element will return the same array positions. If any of these returned positions is 0 in the bit array, then we can conclude that the queried element was not present in the original set. If all n positions are 1, then there are two possibilities: either the element was present in the original set, or those bits are set to 1 due to insertion of other elements, and their combination is by chance. The latter case means that Bloom filter will return a false positive by indicating that the queried element was in the original set, while it was not. The false positive rate of a Bloom filter can be calculated based on *m* and *n*, and *n* can be optimized for a desired false positive rate

The main advantage of Bloom filters is a significant reduction in memory requirements of a lookup operation. Rather than the entire queried set, only a bit array of fixed size needs to be stored in the memory. Furthermore, since there are no false negative results, sensitivity (or recall rate) is 100%. The trade-off is the computational cost of hashing and probability of false positives. Due to its advantages, Bloom filters have been used to store sequences for use in fast,

specific, and sensitive biological sequence classification (Chu, et al., 2014; Stranneheim, et al., 2010).

### 2.1.2 Biobloom tools

Chu *et al*'s BBT is a Bloom filter implementation that is specifically designed for sequence categorization. For this application, input sequences are broken down to k-mers, which are then inserted into a Bloom filter. When querying, the query sequence is divided into k-mers of the same length, which are then checked for their presence in the Bloom filter (Figure 2.2). In addition to its lower memory requirements, BBT is faster than two industry-standard alignment tools, **B**urrows-**W**heeler **A**ligner or BWA (Li and Durbin, 2009) and Bowtie2 (Langmead and Salzberg, 2012) while retaining comparable level of accuracy. This is achieved by including heuristics to control for false positive rate and increase speed. Following is a more detailed description of the BBT pipeline.

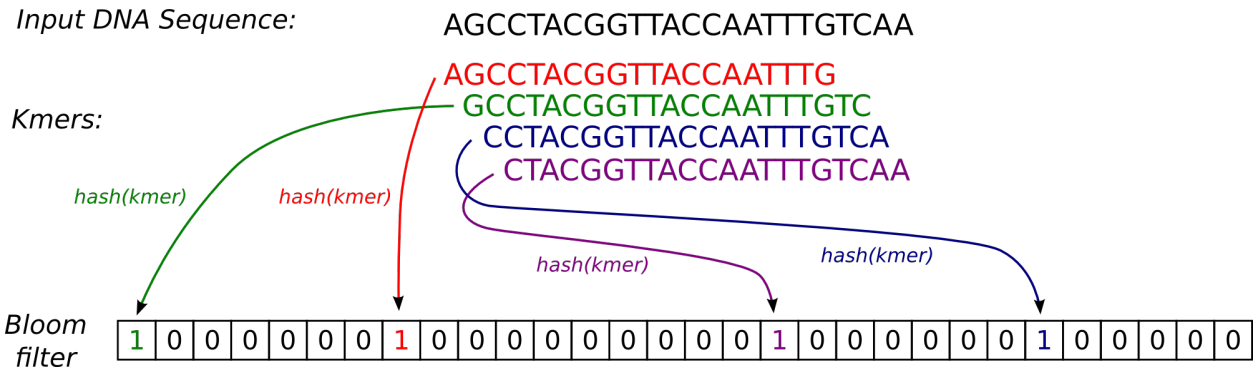



**Figure 2.2 Application of Bloom Filters to Sequence Categorization**

Using hash functions, k-mers generated from the input sequence are deterministically mapped to the specific locations in the memory, which are set to 1. An identical sequence can be detected with querying this Bloom filter, since using the same value of k and hash functions will return the memory positions that are previously set to 1.

There are two tools in the BBT pipeline. The first tool is biobloommaker, which takes a set of nucleotide sequences as input, extracts all possible substrings of length *k* (k-mers) and inputs them through a fixed number of hash functions. The returned hash values denote the bit signature for each k-mer, and together those bit signatures constitute the Bloom filter. Directionality is handled by comparing forward and reverse complements of every k-mer, and only including the alphanumerically smaller sequence in the filter. Users can input the length *k* and a desired false positive rate for the Bloom filter, which changes number of hash functions used. After the Bloom filter is built and saved, the second tool, biobloomcategorizer can be used to query the Bloom filter with a new sequence, which is also converted into k-mers in a sliding window fashion, i.e. starting from one end of the sequence and shifting one base pair at a time. These k-mers are checked against the Bloom filter one by one and a score is calculated. If a k-mer generated from the query is present in the Bloom filter, the score is incrementally increased according to the formula below:

$$s = \sum_{i=1}^{c} \sum_{j=1}^{a_i} \frac{1 - \frac{1}{(j+1)}}{l-k}$$

Where *c* is the number of contiguous stretches of adjacent filter-matching k-mers until the current position in the query, and $a_i$ is the length of the i-th stretch, while *l* is the length of the query sequence and *k* is the k-mer length. The requirement for contiguous stretches of k-mers penalizes spurious false-positive hits. If the score reaches a user-defined threshold (the *s* parameter), biobloomcategorizer stops processing the query and classifies it as a match. During this process, biobloomcategorizer skips *k* k-mers when a non-matching k-mer is detected after a

long stretch of contiguous hits. This heuristic increases the speed efficiency in cases where a query has only a few mismatches with respect to the compared reference sequence.

### 2.1.3 Progressive Bloom filters

Progressive Bloom filter is a novel implementation of Bloom filter data structure that is provided as a part of BBT. It is based on the idea that, as long as the false positive rate is kept sufficiently low (by setting the size of the Bloom filter and the number of hash functions used), a Bloom filter can be expanded by adding more k-mers during the querying of a large database.

Progressive Bloom filters use the user-inputted seed sequences to initially populate the filter, and then scan a read set. In the Kollector pipeline, users provide a list of files with the one containing seed sequences as the first one, followed by the genomic read files. Once the seeding is done, reads are compared against Bloom filter one by one, and any matching read (based on the *r* parameter, calculated the same way as the *s* parameter) is immediately k-merized and added to the Bloom filter. In the Kollector pipeline, *r* parameter is used during the tagging phase, hence it is defined as a separate parameter from *s*, which is used in the recruiting phase. It should be noted that any *r* threshold less than 1 requires only partial sequence identity between read and the filter for categorization as a match. Thus novel k-mers can be added to the filter from matching reads. During the scanning of the genomic reads, the progressive Bloom filter will add more and more k-mer content, allowing one to incorporate (tag) novel sequences not found in the original set of seed sequences (Figure 2.3). These tagged sequences will overlap with and extend the seed sequences. Read tagging process continues until the Bloom filter reaches a user-defined maximum number of *k*-mers (*n* parameter) or until all the genomic reads are processed.

Progressive Bloom filter implementation also includes a heuristic to take advantage of paired-end nature of most Illumina datasets. The method works by processing read pairs rather

than single reads. When adding a read pair, Kollector uses slightly different criteria for a read pair to be considered a match between different stages. In the tagging stage, a read pair is added to the filter if one of its reads is matched. This increases the sensitivity of the tagging process, as it allows for incorporation of more k-mer content. In the recruiting stage, both reads have to match for the pair to be recruited, thus increasing the specificity of the recruited sequences with respect to filter.

### 2.1.4 Progressive Bloom filters in the context of Kollector

In the Kollector pipeline progressive Bloom filters are initialized with transcript sequences, and intronic regions are incorporated into the progressive Bloom filter by scanning genomic reads. An illustration of this process for single *C. elegans* transcript *C17E4.10* is shown in Fig. 2.3. For generating this figure, all tagged reads were outputted in chronological order, and aligned to the reference genome. As one would expect, initially tagged reads overlap exonic regions totally or partially, but as those reads are added to the filter they incrementally allow reads from intronic regions to be tagged as well. When compared with raw coverage of the genomic reads, there are dips in coverage in the middle of longer introns for the tagged reads. This is presumably because these regions are incorporated into the filter later during the run, when most of the reads are already processed. It also indicates that the order of the reads in the input file can affect whether or not these regions will be tagged, as presumably the coverage can drop to zero in the middle of long introns. However, Figure 2.3 also shows that once the entire genic region is tagged, the coverage difference between exons and introns can be mitigated after the second stage of the Kollector pipeline (recruitment, which is described below), as the coverage of the recruited reads is almost equal to the raw coverage.

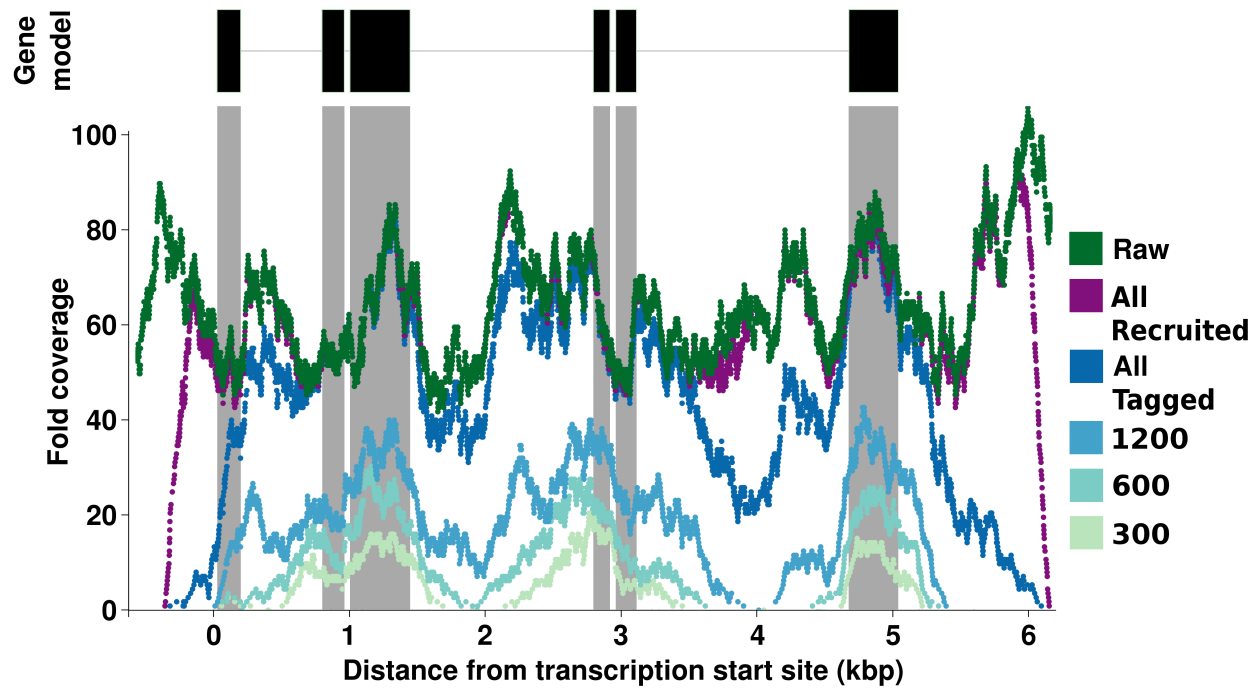



**Figure 2.3 Read Tagging with Progressive Bloom Filters**

Fold coverage of tagged reads is shown at points in time (first 300, 600 and 1200 reads, and all tagged reads) during a single run with the *C. elegans* C17E4.10 gene as target. The raw read coverage is indicated solely as a baseline, to show the tagging process of the progressive Bloom filter. In the Gene Model track, the black rectangles depict the exons, and the connecting grey line depicts the introns.

## 2.2 Read recruitment and *de novo* assembly

In the second stage of Kollector, Bloom filter from the first stage is used with biobloomcategorizer to recruit read pairs that match with the filter (based on the *s* parameter), thus processing the whole genomic dataset once more. In order to increase the sequence specificity of recruited reads, Kollector requires both reads to match for recruiting a read pair.

The recruited reads are outputted in the fastq format and inputted into ABySS for *de novo* assembly in the third stage.

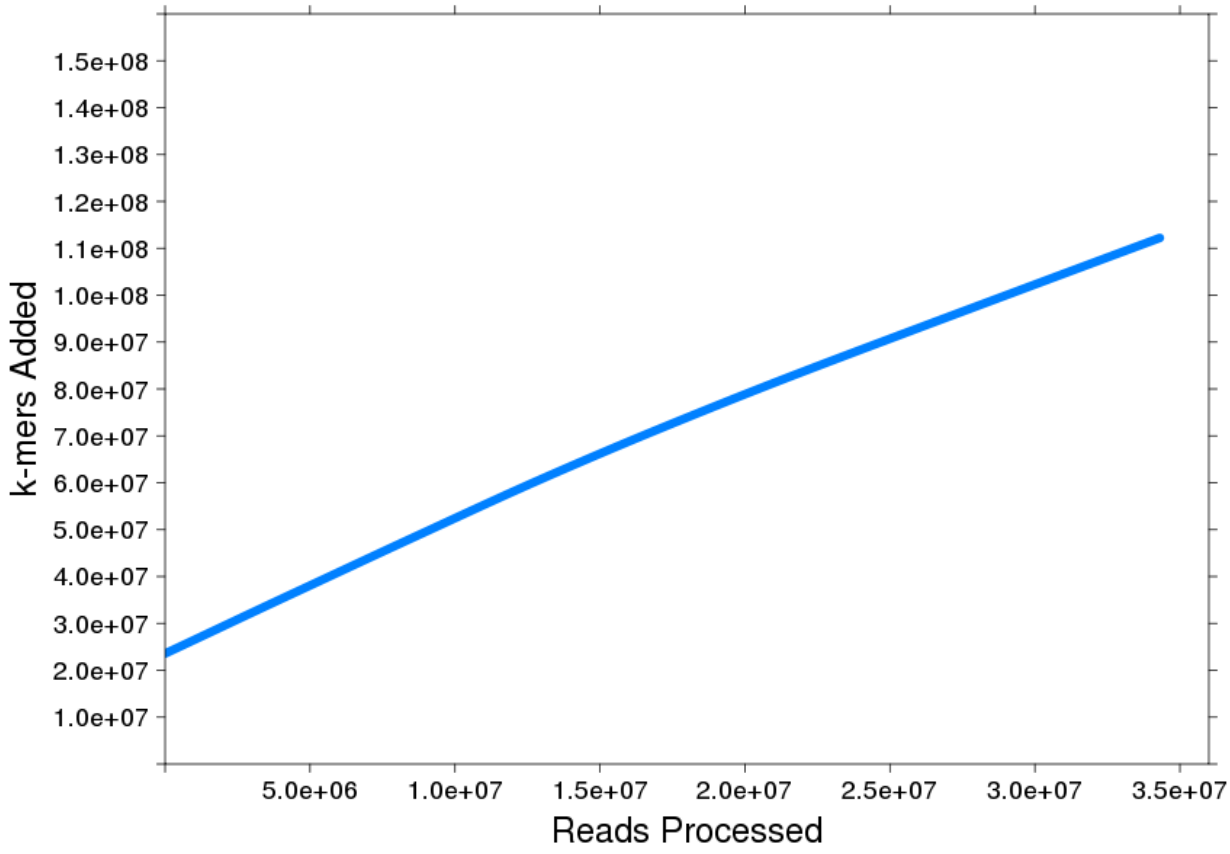



**Figure 2.4 K-mer Accumulation Trend of Progressive Bloom Filters**

K-mer accumulation during a Kollector run on *C.elegans* transcripts demonstrates the greedy approach of progressive Bloom filters. In the absence of a k-mer cap, progressive Bloom filter continued to expand until all reads are processed and acculumated k-mer space slightly larger than entire *C.elegans* genome.

### 2.2.1 *De novo* assembly with ABySS

ABySS (Assembly By Short Sequences) is an assembly tool developed by Simpson *et al* for application to very large datasets produced by NGS (Simpson, et al., 2009). As a short read

assembler, ABySS is based de Bruijn graph paradigm. In this approach, all possible substrings of length k, or k-mers, are generated from the sequence reads. K-mers that have k-1 nucleotides of sequence overlap are then linked together by a de Bruijn graph. This graph is then traversed to reconstruct the target sequence (Figure 2.5). The principal advantage of ABySS is that its de Bruijn graph need not to be stored in a single computer, it can be distributed over a network of computers, or nodes. This distributed representation of de Bruijn allows the assembly algorithm to run parallel computations in several nodes. This parallelization in turn enables processing and assembly of large NGS datasets, such as those from eukaryotic organisms, in a timely manner. Following is a more detailed description of the ABySS algorithm.

ABySS first loads reads into the memory by breaking each of them into k-mers. From each input read of length l, (l-k+1) k-mers are generated in by sliding a window of length k over the read. A de Bruijn graph is built in which each k-mer is a vertex and a k-1 overlap between two adjacent k-mers is represented by an edge. In this graph, read errors cause death ends, or tips. So ABySS algorithm prunes these tips to remove erroneous reads from the assembly. Another issue that has to be dealt with is sequence variation in the input data. Variant sequences, or repeat sequences with small differences cause two or more local divergent paths to emerge and form a bubble, thus creating ambiguity in the graph. Hence, ABySS removes those variant sequences from the assembly by popping the bubble, *i.e* resolving the ambiguity in favor of single branch, typically the one with the higher k-mer coverage, and removing the others. After these steps, vertices that are linked by an unambiguous path of edges are merged, creating the uniquely assembleable contigs or unitigs. This concludes the first stage of the assembly, which is

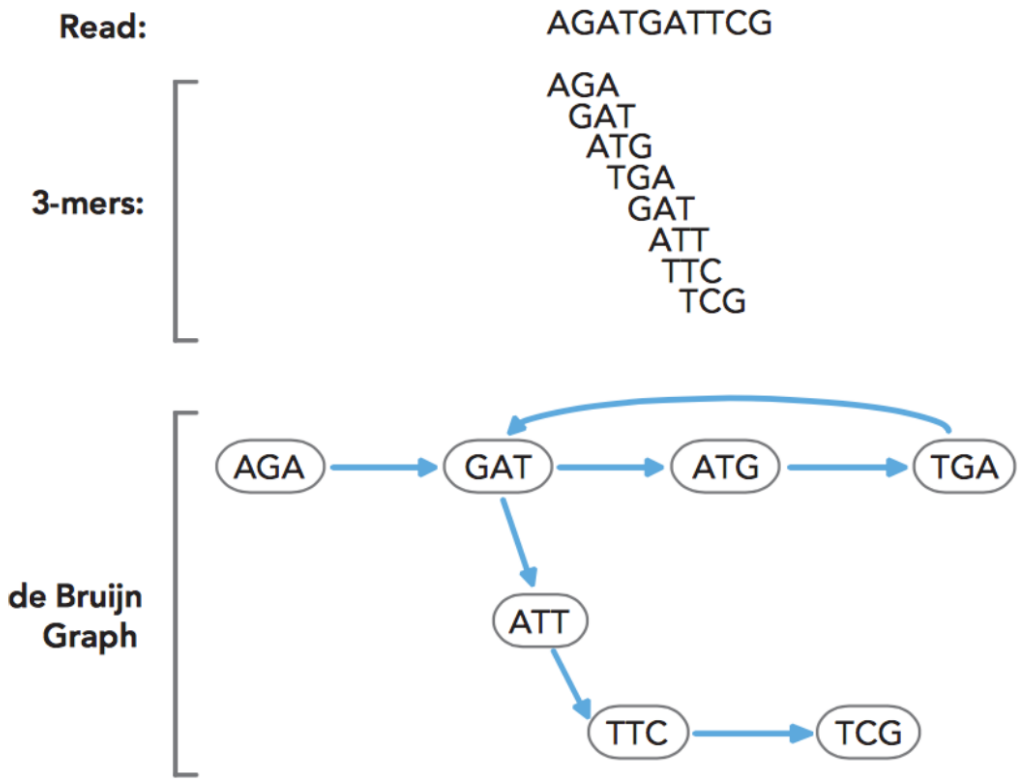



**Figure 2.5 Using de Bruijn Graph for Assembly**

Application of the de Bruijn graph to the assembly problem, with an example k value of 3. Input read is broken down to 3-mers, and a graph is built using the 3-mers that overlap by 2 nucleotides. Once the graph is built, an assembly algorithm can traverse the vertices using the illustrated path and reconstruct the original sequence.

also called the single end or unitig assembly, which stands uniquely assemble-able contigs. In the second stage, paired-end reads are aligned to the assembled unitigs, and an empirical distribution of the distance between two paired reads is obtained. Using this distribution and the paired reads

which are aligned to different unitigs, ABySS estimates the distance between unitigs, and generates an adjacency graph of linked unitigs. It then finds the unambiguous paths along the adjacency graph that agree with the distance estimates, and merges the unitigs along these paths to create contigs. This is the paired-end, or the contig stage of the assembly. In the final stage, ABySS aligns sequences with more linkage information (input transcripts, in Kollector's case) to the contigs and generates an adjacency graph based on contigs that aligned to the same transcript. In this adjacency graph, contigs that are linked on an unambiguous path are merged into longer sequences called scaffolds. Kollector pipeline uses these scaffolds for post-processing.

### 2.3 Post-processing

The main goal of the post-processing step is to identify and report scaffolds that contain the target genes. This is achieved by aligning the input transcripts to assembled scaffolds using the **G**enomic **M**apping and **A**lignment **P**rogram or GMAP (Wu and Watanabe, 2005), which is an aligner specifically designed for mapping cDNA sequences to the genome.

GMAP optimizes cDNA to genome alignments by identifying splice sites and allowing large gaps in between splice sites, corresponding to introns. One advantage of GMAP is that it does not rely on pre-computed probabilistic models of splice sites for detection. Rather, cDNA is first mapped to the genome to get an overall approximate alignment. GMAP then re-computes the alignment in regions surrounding potential introns with a specialized algorithm. This algorithm includes several heuristics, such as reduced gap penalties for codon insertions or deletions. The absence of splice site models, which are usually generated from model organism data, in this process means that GMAP is generalizable across many different species. This is an important consideration for Kollector, as it aimed at applications involving non-model organisms. Another advantage is that GMAP-reported alignments reflect the given data rather

than prior probabilities. This enables GMAP to generate fairly accurate gene structures. Wu and Watanabe (2005) tested GMAP on a dataset of 882 full-length *H. sapiens* mRNAs with computationally induced mutations at a rate of 3 %. They observed that 99.3 % of the splice sites in the GMAP alignments agreed with Ensembl annotation, the gold standard used in their evaluation. This high accuracy makes GMAP suitable evaluating Kollector results. Lastly, GMAP can align a single cDNA to a human-sized genome in about a second, using as little as 128 MB RAM and with the possibility of multi-threading to handle increased numbers of input. Therefore, the GMAP alignment step often does not constitute a bottleneck in Kollector pipeline when handling large eukaryotic genomes in reasonable time.

After the GMAP alignment, Kollector parses the results in the sequence-alignment/map (SAM) file. It calculates a percent sequence identity as well as a percent query coverage, which is defined as number of matched nucleotides over the total length of input transcript, for each alignment. Kollector then reports the alignments that have sequence identity and query coverage scores above a threshold (by default, 90 % for both) as successful. The scaffolds that are successfully aligned to input transcripts are extracted from the assembly and outputted in a separate fasta file. Figure 2.6 shows a sample from Kollector output, compared to the input transcript and the reference genome. As expected, Kollector assembled gene includes intronic regions, which are missing in the input transcript.

## 2.4 Improving Kollector performance with repeat filtering and read shuffling

Kollector is designed to work with large eukaryotic genomes. One of main challenges posed by these genomes is the relative abundance of low-complexity sequences and repetitive regions. When added to a progressive Bloom filter, these sequences would lead to incorporation of off-target regions due to their sequence similarity to different genomic loci. When incorporated these

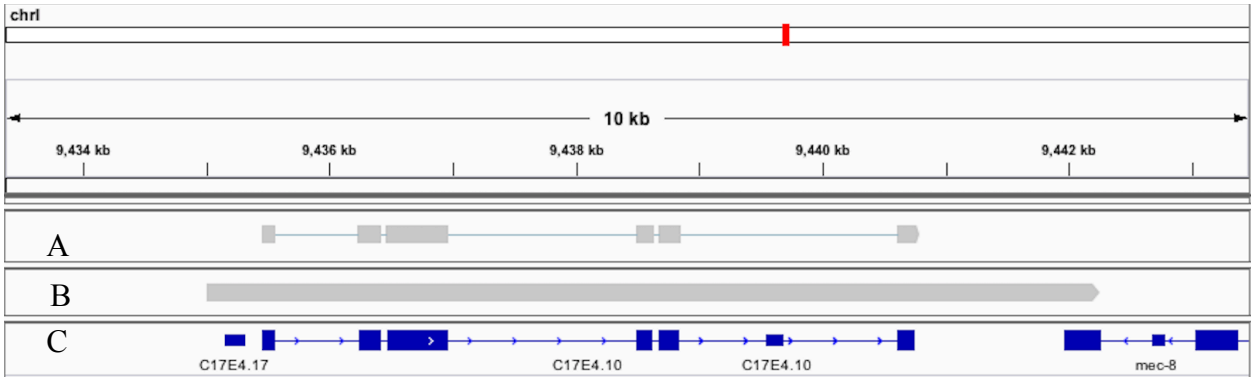



**Figure 2.6 Kollector Output**

Sample Kollector output for *C. elegans* aligned to the reference genome (A), alongside the input transcript (B) and the Ensembl annotated gene models (C). Notice that even though one of exons in the gene model is missing in the input transcript, progressive Bloom filter algorithm was able to recruit reads from these regions as well, and as a result the output encompasses the whole gene.

off-target regions can cause the progressive Bloom filter to reach the k-mer cap before target genic regions are tagged completely. Furthermore, the presence of reads from off-target regions increases the graph complexity during the assembly process, thus reducing the quality of assembled sequences. Not to mention the waste of computational resources that goes to assembling these off-target regions. For these reasons, we have developed a repeat filtering procedure for the tagging phase of Kollector, mainly in order to improve assembly performance in complex genomes.

Biobloommaker can take a Bloom filter of repeat sequences as an additional input (-s option). Repeat sequences that are used to build the repeat filter can be obtained from a repeat

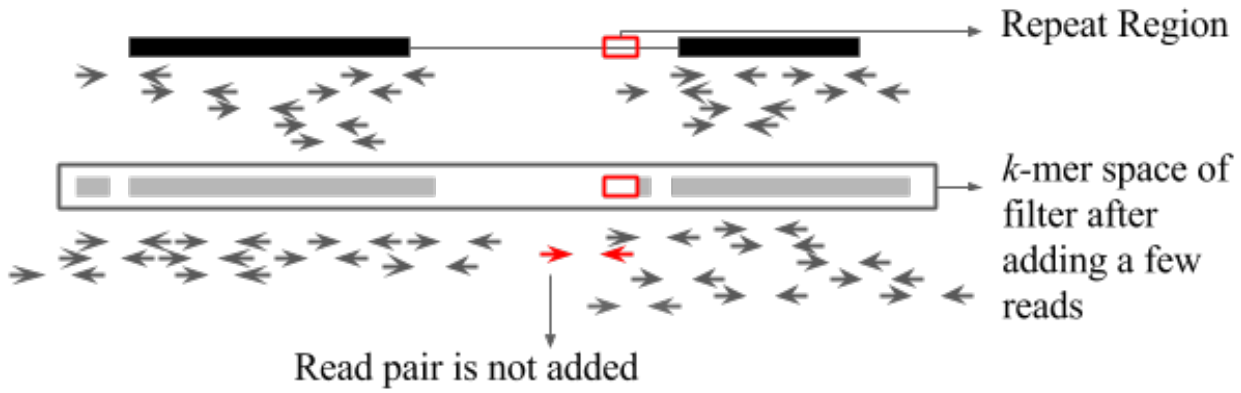



**Figure 2.7 Repeat Filters**

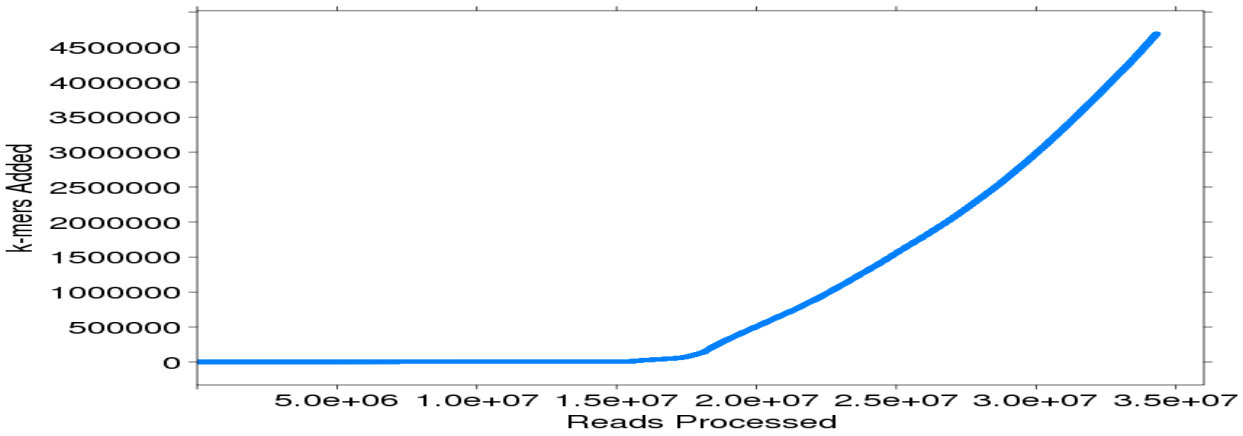
Repeat filtering during the tagging phase. K-mer space denotes sequence content covered by the totality of the k-mers in the Bloom filter. Black bars show depict the k-mer space when the Bloom filter is imitated with the input transcripts. The grey bars depict the k-mer space after read tagging is started. When a read, shown here as arrows, overlaps with a repeat region, it cannot be used for tagging new reads.

database, or by recording k-mer multiplicities in input genomic reads and selecting those that have counts above a certain threshold. When adding new k-mers to progressive Bloom filters, they are checked against the repeat Bloom filter, and marked as repeats if they match. Those marked k-mers are added to the progressive Bloom filter, but they are not used for tagging new sequences, thus preventing the recruitment of off-target regions (Figure 2.7).

In order to observe the effects of repeat filtering, we performed Kollector runs for a single transcript with and without using Bloom filters. Figure 2.8 shows the differing k-mer accumulation patterns. In the absence of a repeat filter, progressive Bloom filter goes through a exponential recruitment phase, most likely induced by the incorporation of repeats and other off-target regions. As a result, the ABySS assembly produced by this Kollector run contains many small contigs in addition to target gene, with a total reconstruction 500 kbp. Conversely, when a repeat filter is used, the exponential recruitment phase does not occur, and the resulting assembly

contains a single contig of 8 kbp, which represents the target gene. The limiting of recruited k-mer space also improves the reconstruction performance of Kollector. On a set of 4,500 *Rana*

**A**




**B**

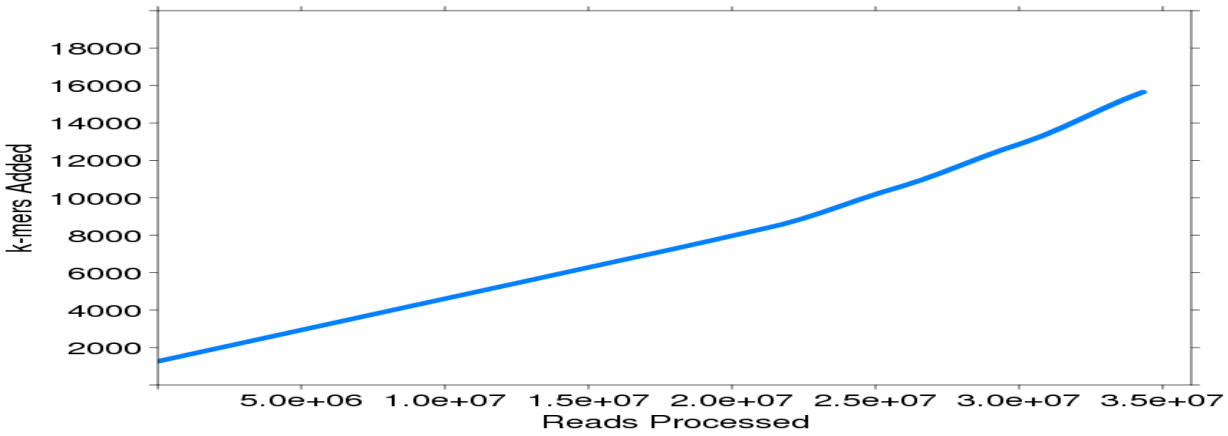



**Figure 2.8 Repeat Filters and K-mer Accumulation**

K-mer accumulation for single transcript progressive Bloom filter in the absence (**A**) and presence (**B**) of repeat filter in *C. elegans,* which suggest that exponential growth phase might be induced by repeats.

*catesbeiana* transcripts, usage of the repeat filter increases the number of successfully assembled transcripts from 403 to 1,820. Taken together, these results demonstrate the utility of repeat filters in preventing the recruitment of off-target regions and improving the performance.

Another feature of the Kollector pipeline that improves the performance is the option to shuffle the genomic input reads before the tagging stage. Progressive Bloom filters are sensitive to the order of reads in the input file. This is caused by the greedy nature of the recruitment algorithm. For instance, it is more optimal if reads containing intron-exon boundaries are at the top of the file, since they can be added to the pBF early in the run, which makes it more likely to recruit intronic reads later on. On the other hand, if those regions are closer to the bottom of the file, certain introns may not be recruited. The read shuffling option aims to reduce such potential bias created by the read order. This is achieved by splitting the input read files into smaller ones containing around 1 million reads each, and then using Bash's random sorting function to change the order of the file list that is fed into Kollector. Therefore, targets that are failed to assemble in a Kollector run can be re-tried in an another run with a different read order using this option.

### 2.5 Implementation

Kollector is implemented using Bash (https://www.gnu.org/software/bash/), the command-line interpreter for GNU, a Unix-like operating system. Kollector script is freely available for non-commercial use at https://github.com/bcgsc/kollector.

# Chapter 3: Experiments

This chapter describes a series of experiments performed to evaluate the Kollector pipeline and showcase several of its possible applications. **Section 1.1** describes the results of Kollector runs on *H. sapiens* and *C. elegans* datasets, including quality control of the assemblies and an analysis of failed cases. **Section 1.2** shows the comparison of Kollector to the current state-of-the-art. **Section 1.3** demonstrates several use cases for Kollector. These cases include comparative genomics, genome-scale targeted gene assembly, sequence detection and improving a draft genome with Kollector output.

## 3.1 Performance of Kollector in model organisms

We ran Kollector on publicly available *H. sapiens* and *C. elegans* datasets (Table 3.1). Since

**Table 3.1 Datasets used in targeted assembly experiments**

| #Species | Read Lengths | Total Bases | Raw Cov. | Source | Bait |
|---|---|---|---|---|---|
| *C. elegans* | 110bp | 7.5Gbp | 75x | SRA Accession: DRR008444 | 13,556 Refseq mRNA sequences (>1kb) |
| *H. sapiens* | 250bp | 229Gbp | 70x | SRA Accession: ERR309932 | Trans-ABySS (v1.5.1; $k$ = 42) assembly of 52,006 transcripts from TCGA dataset barcode: 22-4593-01 |
| *P. glauca* | 150-300bp | 1.2Tbp | 48x | SRA Accession: SRR1982100 | 16,386 high-confidence transcript models from Genome Annotation: GCA_000966675.1 |
| *P. schaeffi* | 100bp | 12.8Gbp | 128x | SRA Accession: SRX390495 | 1,534 conserved *P. humanus* cDNA sequences Dryad DOI: http://dx.doi.org/10.5061/dryad.9fk1s |
| *M. musculus* | 150bp | 116Gbp | 41x | SRA Accession: SRX1595526 | 4,025 *H. sapiens* orthologous transcripts from Emsembl bioMART |
| *H. sapiens* | 100bp | 13.8Gbp | 4x | TCGA barcode: TCGA-BA-4077 (subset) | HPV type-16 reference genome from Papillomavirus Episteme |
| *R. catesbeiana* | 150-250bp | 382Gbp | 66x | BioProject ID: PRJNA285814 | 794,291 transcripts from Bullfrog Annotated Reference Transcriptome (BART) |

these model organisms have high-quality reference genomes available, we were able to assess Kollector's performance thoroughly by evaluating and characterizing the resulting assemblies.

### 3.1.1 Kollector assemblies

Before running Kollector on *H. sapiens* we randomly divided the transcriptome (Table 3.1) into five bins of 10,000 transcripts each. Each bin is used to initiate a single progressive Bloom filter in a separate Kollector run. We do the binning to prevent pBF from reaching the k-mer cap (n=100,000,000) prematurely, which is before incorporating all of the target gene sequences. The number of the transcripts in each bin is estimated based on the *n* parameter and average human gene length, which is around 10 kilobase pairs (Sakharkar, et al., 2004). Kollector performs incrementally better when used iteratively. In this mode, genes that are not assembled in initial stage are provided as input for the next iteration and re-tried with different parameters. After each iteration, the target transcript sequences are aligned to the Kollector output with GMAP (Wu and Watanabe, 2005), and those that have a perfect match to a single genomic contig along a certain fraction of their length (default = 90%) are deemed to have been successfully reconstructed. Transcripts that do not satisfy this criterion are re-tried in the subsequent iteration by setting a lower specificity for sequence selection during the read collection phase. This is achieved by lowering the r parameter in each iteration (e.g. 0.90, 0.70, 0.50, 0.30, 0.20), while keeping the other parameters constant (s = 0.50, n = 100,000,000 in our experiments). In addition to that, failed transcripts are re-binned before each iteration with the same number of bins, in order to reduce any bias that might be introduced by the binning. For *C. elegans* transcripts, we used the same parameters but skipped the binning step, as number of transcripts was low (Table 1).

In our tests, we ran 10 independent experiments with the read shuffling option, and compared them to 10 independent experiments that are ran without this option (Figure 3.1). We ran the unshuffled experiment several times to observe the randomizing effect of using multiple threads.

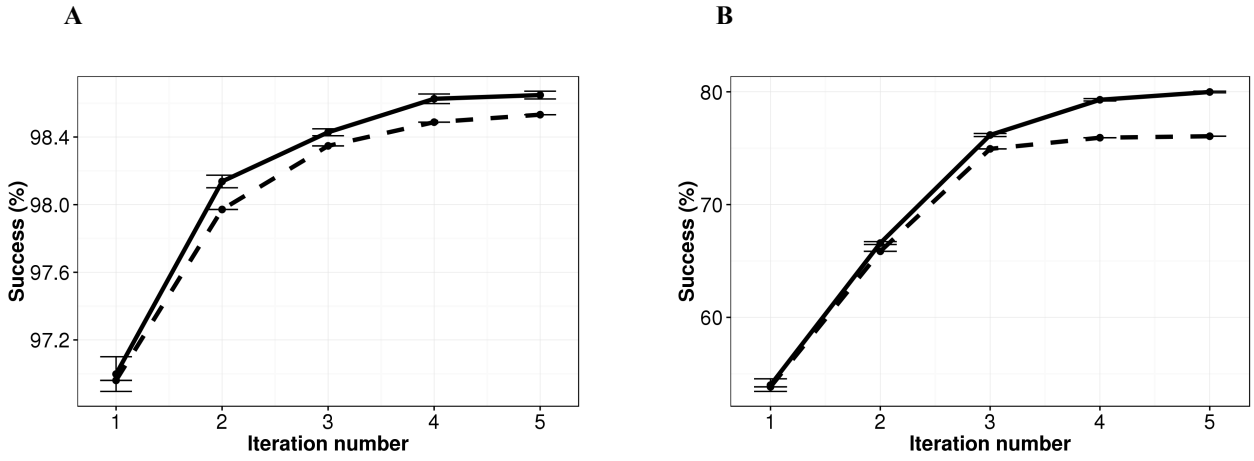



**Figure 3.1 Performance of Kollector for assembling target sequences in (A) *C. elegans* and (B) *H. sapiens.*** Genomic reads were randomly shuffled before each iteration (solid lines, mean of 10 independent experiments). For comparison, runs with no shuffling are also depicted (dashed lines, mean of 10 independent experiments). Error bars denote standard deviation. The success rate was calculated as the cumulative proportion of target genes successfully assembled over five iterations.

In this mode, the exact order that reads are processed by BBT can change slightly from run to run, due to reads being assigned to different threads during the scanning phase. However, this shuffling effect is localized as it occurs within batches of approximately 100 reads, so compared to Kollector's global shuffling option, its effects are quite limited and as expected, produced little variance in the success rate between independent unshuffled experiments (standard deviation < 0.0001 % in both species). On the other hand, Kollector's shuffling option led to statistically significant gains (determined by the Wilcoxon rank-sum test) in the overall assembly success

rate for both species, reaching 98.7% in *C. elegans* (p-value=0.00017) and 80.1% (0.00017) in *H. sapiens*. These results also indicate that the method can still fail to assemble certain genes. Properties of these failed cases in *C. elegans* are analyzed in the next section.

### 3.1.2 Charecterization of failed cases

One of our aims for running Kollector on model organisms was to characterize sequence features of genes that failed to assemble. Since our *C. elegans* transcript set has been annotated by NCBI Refseq with the gene names, we are able to obtain high-quality full sequences of all target genes. Using these sequences, we investigated the failed reconstruction of 199 *C. elegans* transcripts (unshuffled experiment, Fig 3.1A dashed line). By comparing the length distribution of two groups, we found that the missing genes were on average significantly longer than the successfully assembled ones ($p=1.5x10^{-5}$; Student's t-test; Fig. 3.2), indicating that the failure was due in part by the size of the gene. Although the distributions suggest substantial overlap, and there were many long targets with successful assembly, with lengths comparable to the failed targets, our statistical test suggests that Kollector has a bias towards assembling smaller

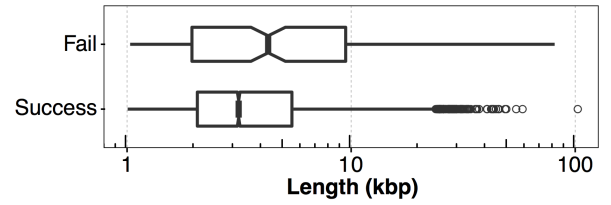



**Figure 3.2 Gene Length Comparison in Failed Cases**

Length comparison between the C. *elegans* target genes that are successfully assembled (bottom) vs. those that failed (top) to assemble with Kollector. Notches in the boxes represent a 95% confidence interval around the median. The length difference between two groups, represented on a logarithmic scale on the x-axis, is found to be statistically significant by t-test ($p=1.5x10^{-5}$).

genes, likely due to the challenge of identifying enough genomic reads to connect exons separated by long introns.

As Fig 2.3 shows, there is a drop in coverage in intronic regions during the tagging phase. The figure also shows that coverage in these regions is recovered in the recruiting stage. However long introns might not be tagged completely if coverage drops to zero inside the intronic sequence, which will result in gaps in the assembly. In order to investigate this possibility, we did a similar analysis to the gene length comparison, but this time we used the the length of longest intron, which is most likely to cause a substantial drop in coverage. We found that failed genes have significantly longer maximum intron lengths than successfully assembled genes ($p=1.5x10^{-8}$; Student's t-test; Fig. 3.3A). For targets with maximum intron lengths of approximately 20 kbp, we expect 50% of the reconstructions to fail reconstruction. However, we note that these large intron genes make-up a very small proportion of the total dataset (Fig. 3.3B). We expect that the use of longer reads, high coverage datasets and and insert sizes (possibly mate-pair reads) could help alleviate some of these issues.

### 3.1.3 Quality control of the Kollector assemblies

To evaluate the accuracy of the genomic contigs produced by Kollector, we aligned the output of a Kollector run from *H. sapiens* and *C. elegans* to their respective reference genomes with BLAST (Altschul, 1990), calling a correct alignment at a threshold of 95% query coverage and 95% sequence identity. Doing so, we found that 99.8% and 99.7% of the assembled genes satisfy these criteria in *C. elegans* and *H. sapiens* respectively (Table 3.2). Small number of Kollector

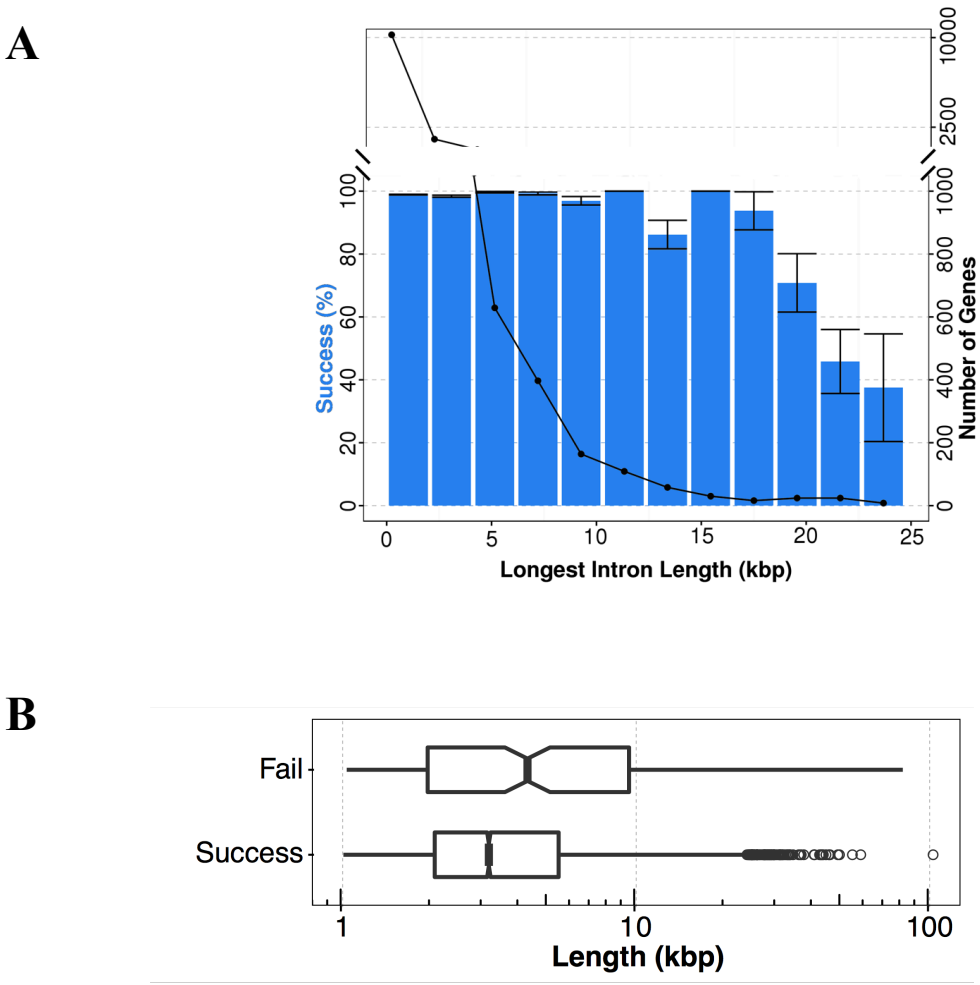



**Figure 3.3 Analyis of Failed Cases**

**A) Proportion of successful gene assemblies vs longest introns (bars) with number of genes in each bin (lines).** The first and second bins make up 10,237 and 1,496 genes respectively, and make up 86% of all genes in the dataset

**B) Longest intron length comparison between the *C. elegans* target genes that are successfully assembled (bottom) vs. those that failed (top).** Notches in the boxes represent a 95% confidence interval around the median. The length difference between two groups is found to be statistically significant by t-test ($p=1.5x10^{-8}$).

assemblies failed the QC in most part due to repeat number variation. Of the 32 *C.elegans* Kollector assemblies that failed QC, 27 had insertions and 5 had deletions in the repeat regions.

**Table 3.2 Accuracy of Kollector-assembled genes**

| Species | Number of Genes Assembled | Number of Genes Aligned | Success Rate |
|---|---|---|---|
| *C. elegans* | 13,378 | 13,356 | 99.8 |
| *H. sapiens* | 41,631 | 1,525 | 99.7 |

### 3.1.4 Off-target reconstruction in Kollector assemblies

ABySS assemblies produced in the Kollector pipeline contain sequences that do not belong to the target genes. These off-target regions are incorporated into progressive Bloom filters due to presence of repeats and low-complexity sequences. After the assembly step, Kollector uses GMAP (Wu and Watanabe, 2005) to select for contigs that come from targeted genes, so these off-target contigs are not reported to the end user. However, they are still of interest from a software development perspective, since computational resources are wasted for assembling these off-target contigs. Furthermore, reads recruited from these regions also increase the complexity of the de Bruijn graph, which leads to less contiguous assemblies overall. Therefore, we decided to quantify the amount of off-target reconstruction.

This analysis was performed on three bins of 10,000 *H. sapiens* transcripts each. First, sequences for target genes were extracted using transcript to reference genome alignments, including 1 kbp overhangs in upstream and downstream of the alignment. After the Kollector run is performed, resulting assemblies were aligned to the target genes with BWA (Li and Durbin, 2009). Contigs that aligned to a target gene with high quality are considered on-target. Multi-mapped contigs were considered as repeats and discarded from the total analysis. Remaining contigs are counted as off-target sequences. The results show that roughly sixty percent of assembled contigs are off-target, and their combined length corresponds to forty percent of total reconstruction (Table 3.3). These proportions, which are not inconsiderable, ideally can be reduced by more stringent

parametrization and repeat filtering. However, future development of Kollector pipeline, such as a method to determine target gene borders and limit read recruitment, can also be immensely useful for alleviating this problem.

**Table 3.3 Off-target reconstruction**

| Bin | Number of Contigs on Target>500 bp | Length of Contigs on Target (Mbp) |
|---|---|---|
| 1 | 8617 (36.49%) | 31.6 (48.79) |
| 2 | 10544 (42.70%) | 42.9 (59.36) |
| 3 | 10581 (42.00%) | 42.6 (58.97) |

## 3.2 Comparison of Kollector with other targeted assemblers

### 3.2.1 Comparison to aTRAM using cross-species assembly

We assessed Kollector's performance against aTRAM (Allen, et al., 2015), which to our knowledge, is the only tool designed to assemble genes guided by an input protein sequence. This is achieved by an iterative pipeline, in which reads are recruited based on an alignment to the bait sequence. These reads are then assembled and used as the bait in the next iteration, thus incorporating sequences that are missing in the original input.

In their study, Allen and co-workers (2015) used a dataset of 1,534 conserved proteins from human head lice *(Pediculus humanus*) (Johnson, et al., 2014). These were obtained by comparing sequence divergence rates between human, chimpanzee and *P. humanus* orthologous genes using pairwise alignment performed by MUSCLE (Edgar, 2004). This set of protein sequences was used as the bait to assemble orthologous genes in the chimpanzee lice (*Pediculus schaeffi*) genome. It is estimated that these two species have diverged 5.6 million years ago

(Reed, et al., 2004). For comparative purposes it is worth noting that their host species, humans and chimpanzees, have begun diverging around 13 million years ago (Venn, et al., 2014).

For comparison with aTRAM, we ran Kollector using the same whole genome shotgun reads and corresponding cDNA sequences for each orthologous gene, since Kollector cannot process amino acid sequences as input. For the aTRAM itself, we used protein sequences as input for best performance, since more conserved sequences are preferred for optimal read recruitment (Personal correspondence with Julie M. Allen). We compared the results from both tools using two metrics: length proportion of the target gene that aligned to the assembled scaffold with BLAST, and number of assembled genes that passed a reciprocal best BLAST hit test with each target gene. Passing the reciprocal BLAST test requires two sequences to be each other's best hit when aligned as a query.

In our tests, Kollector slightly outperformed aTRAM on both metrics. For 1,543 targets, Kollector's average aligned length proportion was 99.3% to aTRAM's 98.6%, and 1,552 genes passing the reciprocal BLAST test compared with 1,530 in aTRAM (Table 3.4). We'd like to note that Kollector outperformed even though it was using cDNA sequences, as opposed to aTRAM's more conserved protein input. We suspect our method is more robust in capturing divergent sequences because our progressive filtering algorithm recruits reads from conserved regions first and then uses them to extend the sequence in possibly more divergent regions. Furthermore, reconstruction of divergent sequences can benefit from using a low *r* parameter (the specificity required for read tagging), which may be preferable if evolutionary distance between two species is considerable. aTRAM, on the other hand, does not have a comparable specificity parameter that can be optimized for divergent sequences. Also, in an iterative approach like aTRAM's, repeat regions and off-target sequences can recruit more reads in each

iteration, increasing the complexity of the graph. Kollector's greedy approach reduces or eliminates the need for iterations, and also allows for repeat filters to prevent recruitment by such regions.

**Table 3.4 Comparison with aTRAM**

| Method | Proportion | Reciprocal Best-BLASTn | Time (h) |
|---|---|---|---|
| Kollector | 99.3 | 1,532 | 2 |
| aTRAM | 98.6 | 1,530 | 25 |

Kollector achieved this task in less than one tenth of aTRAM's run time. The markedly greater speed of Kollector is mainly due to its use of alignment free classification with k-mers and Bloom filters, which allows it to process thousands of transcripts in a single run, whereas aTRAM relies on iterative alignments to individual targets to recruit enough reads for their assembly. For applications on this relatively small scale, the progressive read filtering algorithm used by Kollector eliminates the need for iterations, thus saving time.

### 3.2.2 Comparisons against other targeted assemblers

In addition to aTRAM, we have compared the performance of Kollector on the *C. elegans* dataset **(**Table 3.1**)** to a set of published targeted assembly tools. Although not originally and specifically designed for reconstructing genomic loci from transcript sequences, we tested Mapsembler2 (v2.2.4) and TASR (v1.5.1), since they are the most established and widely used targeted assembly tools available. For instance, TASR is used for structural variant detection in human health research (Brown, et al., 2014; Warren, et al., 2012). For testing these tools, we used the full set of 13,556 *C. elegans* transcripts as input, as well as the genomic read set. For Mapsembler2, we used the default k-mer size (k=31) and the consensus sequence mode (i=2),

which means extensions of the target sequence are collapsed. We ran TASR using the independent, de novo assembly mode (-i 1) with default parameters, meaning targets are only used for read recruitment and not for seeding the assembly. We evaluated results of these tools with the same GMAP alignment criteria that we reported for Kollector. In our tests, Mapsembler2 and TASR assembled 28% and 18% of the targets, respectively (Table 3.5). In contrast, Kollector was able to assemble 97% of the target gene set in the first iteration alone. This is mostly due to the fact that Mapsembler2 and TASR are designed for re-assembling input targets within their respective sequence boundaries, with only limited extension capability in the flanking regions, which limits their utility beyond single exon genes. Kollector's progressive filtering algorithm, unlike the latter approaches, can incorporate reads derived from intronic regions for assembly.

We also chose to compare Kollector against aTRAM and GRAbB, targeted assembly tools that can theoretically assemble the target sequence in full when given an incomplete or partial input. However, both tools require multiple iterations for achieving such an assembly, and we found out that their runtimes become intractable when the number of targets is increased. Therefore, we had to resort to using a random subset of 1000 *C. elegans* transcripts. Another important consideration was the choosing of finishing criteria for the iterative runs. For aTRAM, we used the default finishing criteria, so the is terminated after five iterations. In the case of GRAbB, users can choose between several parameters and one of them is the length of longest assembled contig. We set this parameter to the length of the NCBI Refseq gene for each target. This was done to ensure that GRAbB did not stop iterations prematurely. However, GRAbB was not able to assemble all the targets in a tractable amount of runtime. Therefore, we stopped the run after 48 hours and used sequences that were assembled up to that point for comparison. The

evaluation of these results was done by GMAP alignment, as described above. We found that on this transcript subset, aTRAM and GRAbB assembled 73.1% and 42.5% of the targets respectively, compared to the Kollector's 94.5% success rate (Table 3.5). Furthermore, Kollector substantially outperformed both tools in speed and memory, proving the utility of progressive Bloom filters.

**Table 3.5 Comparison with other targeted assembly tools**

| Method | Number of Targets Attempted | Number of Targets Assembled | Percentage of Targets Assembled | Wall Clock Time (h) | Peak Memory (GB) |
|---|---|---|---|---|---|
| Kollector | 13,556 | 13,144 | 96.96 | 2 | 4.8 |
| Mapsembler2 | 13,556 | 3,742 | 27.60 | 2 | 9.3 |
| TASR | 13,556 | 2,418 | 17.83 | 3 | 15.6 |
| Kollector | 1,000 | 945 | 94.5 | 0.5 | 0.66 |
| aTRAM | 1,000 | 731 | 73.1 | 38 | 2.4 |
| GRAbB | 1,000 | 425 | 42.5 | 48* | 3.1 |

* Run terminated by user

### 3.2.3 Comparisons against *de novo* whole genome assembly

As stated before, Kollector's targeted approach offers significant advantages compared to the whole genome *de novo* assembly. In addition to the reduced time-and-memory complexity, we expect Kollector to rescue regions that cannot be reconstructed by whole genome *de novo* assembly due to localization of the assembly problem. We tested this hypothesis by assembling *C. elegans* and *H. sapiens* datasets with ABySS, using the same assembly parameters with the respective Kollector run. Target transcripts that were used to seed the Kollector run were aligned to the whole genome assemblies. Using the same criteria (90% query coverage and 90% sequence identity), the proportion of successfully aligned transcripts was compared between

Kollector and whole genome approaches. In both species, Kollector assembled substantially more genes than whole genome assembly approach (Table 3.6).

**Table 3.6 Comparion between Kollector and whole genome assembly by ABySS**

| Species | Number of Genes Assembled by Kollector | Number of Genes Assembled by ABySS |
|---|---|---|
| *C. elegans* | 13,378 (98.7%) | 11624 (85.7%) |
| *H. sapiens* | 41631 (80.1%) | 37961 (72.9%) |

## 3.3 Applications of Kollector

### 3.3.1 Cross-species assembly using Kollector

Non-model organisms might not have extensive and well-annotated transcriptomes available to researchers. As such sequences are needed as input in Kollector, it can be argued that this limitation decreases Kollector's utility. However, we believe that this is remedied by Kollector's ability to use transcript sequences from one species to reconstruct the genic regions of a related species. Ideally, when used for cross-species assembly, Kollector will start recruiting reads from the most conserved regions, and then use those to incorporate more divergent regions into the progressive Bloom filter.

In order to demonstrate its utility for cross-species assembly, we tested Kollector using *H. sapiens* transcript sequences as bait to assemble orthologous (>70% sequence identity) *Mus musculus* genes (Table 3.1**,** #4). Despite being separated by approximately 90 million years of evolution (http://timetree.org), Kollector was able to assemble 3,295 of 4,025 target genes in a single iteration (r=0.90 k=96), corresponding to an 81.9% success rate, as assessed using the orthologous *M. musculus* transcripts. This shows that Kollector's r and s parameters allow for a certain degree of sequence divergence between seed and the target sequence.

### 3.3.2 Scaling to large genomes

Assembling complex and very large eukaryotic genomes is a computationally demanding process. Therefore, a targeted assembly approach for retrieving biologically important gene sequences remains an attractive option for most researchers. We have tested such a use case for Kollector using *Picea glauca* (white spruce), which has a genome size in the 20 Gbp range (Warren, et al., 2015). For the species, Warren and colleagues derived a high-confidence set of 16,386 gene predictions and associated transcripts. Using these transcript sequences, Kollector was able to reconstruct 13,277 (80.4 %) of the original target genes in a single iteration. It is worth noting that this was achieved by randomly dividing the transcript set into five bins of equal size. The Kollector run for each bin took 24 hours to complete using a maximum 43.3 gigabytes of RAM.

Researchers are often also interested in the regions immediately upstream and downstream of genes, which typically contain promoters and other regulatory elements. Due to the nature of the progressive filtering algorithm, Kollector assemblies may extend into these regions. In order to demonstrate this, we aligned the aforementioned high-quality transcript models to the resulting Kollector assemblies and quantified the amount of sequence upstream and downstream with respect to the transcript. We show that, in addition to a gene's exonic and intronic sequences, Kollector typically reconstructs approximately 1 kbp of sequence beyond the 5' and 3' ends of the target transcript (Fig. 3.4). Such extensions would enable characterization of the regulatory factors and complexes in the proximal promoter of genes of interest by chromatin immunoprecipitation, and would be especially empowering to studies of non-model organisms without available reference genome sequences.

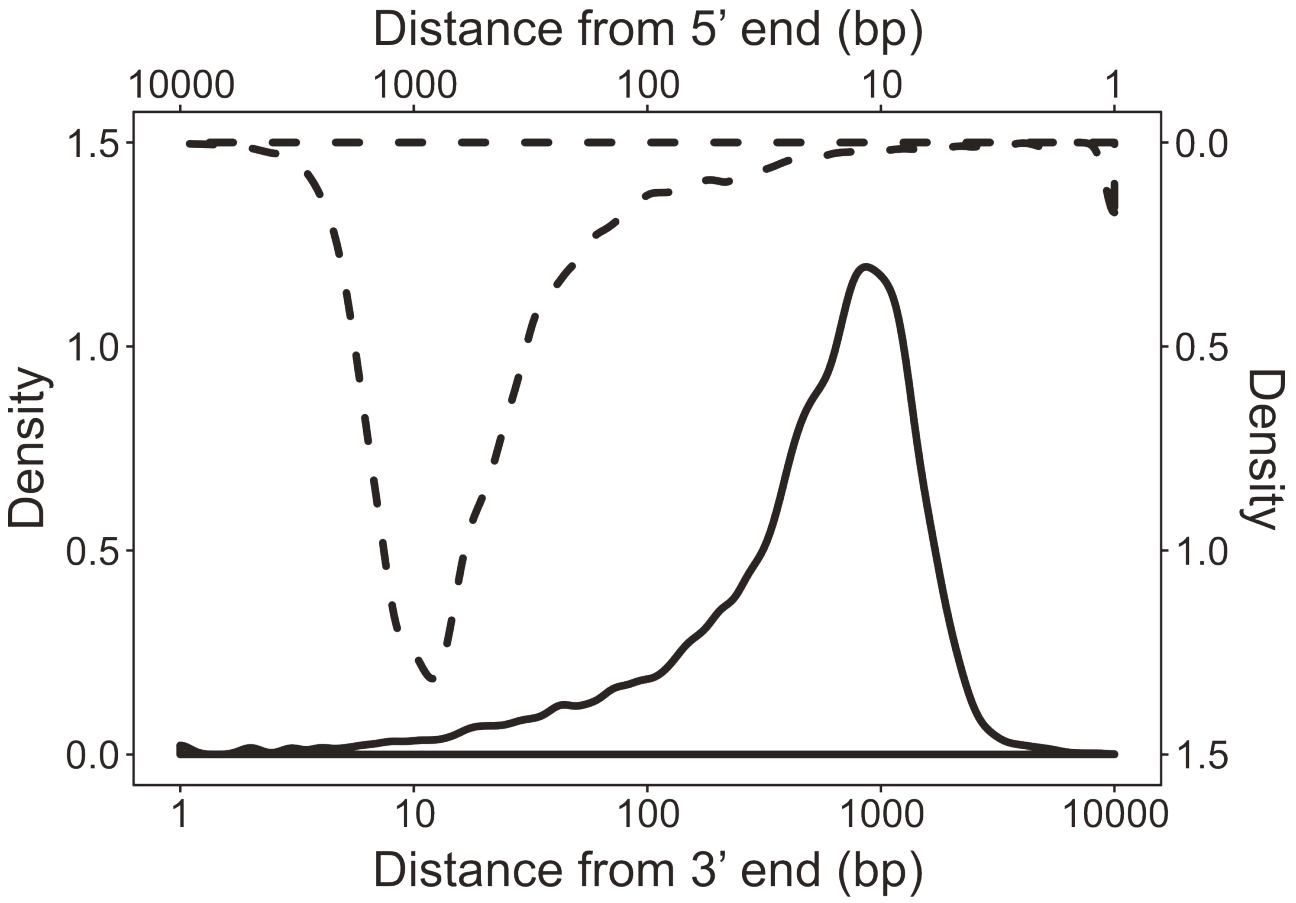



**Figure 3.4 Length distribution of flanking regions, after Kollector assembly of *P. glauca* genes.**

In order to define the flanking regions, we aligned the high-confidence transcript models of white spruce to our Kollector assemblies and quantified the length of the sequence upstream (dashed line, upper x axis and right-side y axis) and the downstream (solid line, lower x axis and left y axis) of the input transcript alignment.

### 3.3.3 Whole genome targeted assembly

A prominent application of de novo assembly is the detection of specific, yet novel, sequences, where a mapping approach can introduce bias, especially for detecting structural variants (Alkan, et al., 2011). However, the large sample size of many studies, such as those of cancer genomics consortia, such as International Cancer Genome Consortium (ICGC) and The Cancer Genome

Atlas (TCGA), can put a strain on computational resources when a *de novo* whole genome assembly is required. Therefore, a fast and reliable targeted assembler, like Kollector, might be an attractive option for researchers, especially when considering its ability to extend incomplete input sequences.

In order to demonstrate the extent of its utility, we have used Kollector for the targeted assembly of Human Papilloma Virus (HPV) in a cancer sample. The Cancer Genome Atlas consortium has profiled 279 head and neck squamous cell carcinomas (HSNCs) and detected many HPV positive samples, which were experimentally confirmed with immunohistochemistry and in situ hybridization (Cancer Genome Atlas, 2015). We ran Kollector on the genomic data from one of the confirmed samples [TCGA-BA-4077] with HPV type -33. Kollector does not need the bait to match the exact target sequence, as demonstrated in Section 1.3.1, so we used HPV type-16 reference genome as bait, and were able to re-assemble the complete HPV type-33 genome sequence in a single iteration with less than 15 minutes runtime and using 1 Gb of memory. We also used genomic reads from the matched normal sample as negative control, and, as expected, Kollector did not yield any assembled HPV sequences. Because Kollector uses only sequences contributed by the reads, the assembled strain remains unbiased relative to our bait sequence.

### 3.3.4 Improving the *Rana catesbeiana* genome assembly

Since Kollector pipeline localizes the assembly problem and reduces its complexity, it can be used to improve genic regions in a genome assembly, which might not be completely reconstructed with the conventional whole genome *de novo* assembly approach. In order to improve the assembly of genes, Kollector takes advantage of corresponding transcriptomic data.

As a proof of concept, we used Kollector to improve the whole genome assembly of *Rana catesbeiana,* or American Bullfrog. Transcript information came from the Bullfrog Annotation Reference Transcriptome (BART) version 2.0. BART was made by collapsing Trans-ABySS (Robertson, et al., 2010) assemblies of RNA-Seq data from five different tissues under different chemical and temperature conditions (Hammond, et al., 2017). BLASTx (Gish and States, 1993) was used to detect highly similar transcripts based end-to-end alignments and only the longest representative for each cluster of similar transcripts is retained. As a result, BART contains 794,291 transcripts that are longer than 500 base pairs, which was the input for the Kollector run. Genomic reads came from Illumina Hiseq PET datasets of 2x150 and 2x250 base pairs length (Table 3.1).

Before the Kollector run, BART is randomly divided into 80 bins of 10,000 transcripts each. Kollector was then run iteratively on each bin and failed transcripts are re-tried in the next iteration with lower $r$ and $k$. After 5 iterations, 78% of BART transcripts were successfully assembled. There are several factors limiting Kollector's success rate in this case. Chief among them is the size and complexity of the Bullfrog genome. More than 60% of the Bullfrog genome is identified as putative repeats (Hammond, et al., 2017). Furthermore, we expect BART transcripts to have mis-assemblies, since they are products of a *de novo* assembly process.

The draft genome we used for the comparison is obtained by assembling the same genomic read set with ABySS (version=1.9.0, k=128). This assembly had an N50 value of 16,865 base pairs and a total reconstruction size of 5.3 Gbp.

In the next step, Kollector assemblies were used to improve the draft genome assembly using ABySS's long scaffolding function. For long scaffolding, ABySS aligns Kollector output

to the draft genome and generates an adjacency graph based on the ABySS contigs that successfully mapped to the same Kollector gene, which are then put together as scaffolds.

In order assess to contribution of Kollector, to the rescue of genic regions, we used Core Eukaryotic Genes Mapping Approach (CEGMA) (Parra, et al., 2007). CEGMA uses a statistical approach called Hidden Markov Model (HMM) to annotate 248 most conserve eukaryotic genes in a given sequence. The number of Core Eukaryotic Genes (CEGs) present in a draft genome was used as a metric to completeness of genic regions. We ran CEGMA on the Kollector output and our draft bullfrog genome, before and after long scaffolding (Table 3.7). These results clearly show that for genic regions, Kollector's targeted approach is more successful in recovering CEGs, compared to whole genome assembly. Furthermore, long scaffolding with Kollector output improves the completeness of the draft genome. This shows for complex eukaryotic genome assemblies, there is value in using Kollector alongside whole genome assembly.

**Table 3.7 Scaffolding of a draft genome with Kollector output**

| Dataset | Complete CEGs | Partial and Complete CEGs |
|---|---|---|
| Kollector Output | 100 | 182 |
| Draft Assembly | 51 | 139 |
| Draft Assembly (scaffolded w/ Kollector) | 65 | 168 |

## Chapter 4: Conclusion

We have presented Kollector, a targeted de novo genome assembly application that takes transcript sequences and whole genome sequencing reads as input, and assembles the corresponding genic loci. Input transcripts are used to seed progressive Bloom filters, a data structure that can efficiently expand into intronic regions using sequence overlaps. Due to our alignment free approach, we demonstrate that Kollector can scale well up to large genomes, such as that of human and white spruce.

We first evaluated the Kollector pipeline on datasets from *H. sapien*s and *C. elegans.* Using their reference genomes, we were able to confirm that Kollector can assemble a vast proportion of its target genes with high accuracy. We note that our *H. sapiens* transcript dataset is a *de novo* transcriptome assembly, so it is similar to the data input we expect to be used for the non-model organisms. On the other hand, our *C. elegans* transcript dataset is from NCBI Refseq database, and since these are annotated, we were able to characterize the failed cases by obtaining the full target gene sequences. We found that target genes that are failed to assemble are longer than those assembled successfully. Furthermore, they also have significantly longer maximum intron length, as determined by Student's t-test. This is consistent with our characterization of progressive Bloom filter recruitment process, which shows a dip in coverage, as the intronic region gets longer.

We then compared Kollector with four published targeted assemblers. Two of them, Mapsembler2 and TASR were chosen because they are well-established tools that are reported widely in the literature. The other two, aTRAM and GRAbB are more recent, but both can be run iteratively to extend an incomplete or partial input sequence. Therefore, they are better positioned to tackle transcript input for genic assembly.

In our comparisons, we showed that Kollector successfully assembled more complete gene loci in significantly less time than aTRAM when assembling the genic space of a related species, and assembles more genes than its four competitors when assembling genes from *C. elegans* transcripts with a significant advantage in time-and-memory complexity. However, we note that this was expected since Mapsembler2 and TASR were not designed specifically for targeted gene loci assembly using RNA transcripts, and aTRAM and Mapsembler2's iterative pipelines do not scale well with increasing number of targets in an eukaryotic genome. Taken together, these comparisons show two main advantages of the Kollector with respect to state-of-the-art: its ability to take full advantage of the transcriptome data and its scalability to large-scale applications in complex eukaryotic genomes.

We also compared performance of Kollector to whole genome *de novo* assemblies performed by ABySS using *C. elegans* and *H. sapiens* datasets. We found that Kollector can reconstruct genes that cannot be rescued by whole genome approach. This proves the utility of localizing the assembly problem with a targeted approach. Furthermore, this result indicates that Kollector output can be of use to improve *de novo* whole genome assemblies of non-model organisms. This can be achieved by scaffolding a draft genome with the Kollector output, as we have demonstrated. Alternatively, Kollector assembled genes can be inputted to a *de novo* assembler alongside genomic reads, so they can be reconstructed in the de Bruijn graph.

After our evaluations and comparisons, we showcased four use cases for Kollector. The first one showed that Kollector was able to effectively assemble gene sequences in *P. glauca* (white spruce), despite its large 20-Gbp genome. These gene assemblies typically included 1 kilobase pairs of upstream and downstream sequence with respect to the input transcript, which could be examined for promoters and other regulatory elements, and enable downstream research

with implications for the study of mechanisms of gene expression. It should be noted that currently the Kollector pipeline does not recognize gene boundaries. In the future, k-mers that mark gene boundaries can be recorded with an additional data structure during the tagging phase. Users can then input a parameter to determine the minimum or maximum number of tagged k-mers beyond gene boundaries, which in turn will affect the length of the flanking regions. As it stands, we already demonstrated that for gene-centric investigations, Kollector could be a robust substitute for *de novo* whole genome assembly, which remains computationally challenging at large scales.

The second use case concerned comparative genomics, where Kollector assembled *M. musculus* genes using orthologous *H. sapiens* transcripts as input. In addition to evolutionary biology, this kind of comparative genomics approach is particularly valuable to researchers working on non-model organisms, which might not have extensive and well-annotated transcript sequences available. This application also indicates Kollector can still perform well in the case of limited sequence identity between the bait and the target, and this feature also formed the basis of our next application.

Our third demonstration involved the detection and whole genome targeted assembly of HPV in a head and neck cancer sample from *H. sapiens*. Kollector's main advantage for this application is that it *de novo* assembles reads of interest, without the risk of introducing artifacts, as typically is the case when aligning reads to a reference genome. Because Kollector can fill in missing sequences by recruiting reads with a progressive Bloom filter, it only requires a limited amount of sequence homology within the bait sequence to fully reassemble a viral sequence. We note that the extent of divergence of the seed sequence that Kollector can use was not fully explored and thus may be interesting to investigate in future studies. Those studies can also

involve using Kollector to detect structural variants that have sequence divergence in different samples. As a result of these studies, parameterization of Kollector can be optimized for more divergent sequences. For instance, *r* threshold can be automatically varied between k-mers that come from the seed sequence and those that recruited later from the reads. Therefore, a divergent seed sequence can be used to recruit reads with a low *r* threshold, while k-mers generated from these reads will have a higher *r* threshold for recruiting other reads, which ensures specificity.

Finally, using the *R. catesbeiana* draft genome assembly, we demonstrated that Kollector output could be used to improve the assembly of genic regions with scaffolding. This is because Kollector localizes and reduces the complexity of the assembly problem, which was also indicated by previous comparison between Kollector and whole genome *de novo* assembly approach. Furthermore, Kollector allows researchers to take advantage of the additional information in the RNA-seq datasets, which is becoming increasingly available for non-model organisms. We should also note the size of *R. catesbeiana* genome, which is estimated to be 5.8 gigabase pairs (Hammond, et al., 2017). This is another testament to the scalability of Kollector for genome-wide, large-scale applications.

In the future, there remain ample opportunities to improve Kollector pipeline and progressive Bloom filters. Algorithm-wise, specificity of progressive Bloom filters can benefit from a more precise stop condition in the tagging stage. Currently, users have to set a k-mer cap rather arbitrarily, based on number of targets and their estimated length in genomic space. However, this rough estimate can lead to sub-optimal results. When set too low, the k-mer cap might cause the tagging step to be terminated before the target gene is completely inserted. On the other hand, setting the k-mer cap too high reduces the advantages of localized assembly, and is a waste of computational resources. So instead of a k-mer cap, a mapping approach can be

employed, where each tagged k-mer is mapped to a target transcript, either directly or via a k-mer that is already mapped to the target. If this information is stored, tagging can be stopped once whole transcript (plus flanking regions, whose length can be determined by the user with a parameter) is mapped to inserted k-mers. As an added benefit, such an approach can track each transcript separately in the case of a multi-transcript progressive Bloom filter, rather than setting a global k-mer cap for all, which is currently the case.

Another feature that can benefit the Kollector pipeline is an option to use the partial assemblies from previous iteration as the input to the next iteration, thus extending the seed sequence. These sort of iterations are already implemented in aTRAM and GRAbB. Ideally, Kollector should not input off-target assemblies for the next iteration. Information from GMAP alignments or ABySS adjacency graph that was generated during the long scaffolding phase can be used select for assemblies that belong to the target region. However, this will most likely require parameterization and further testing. A simpler alternative would be having the progressive Bloom filter scan genomic read files several times, rather than just once, during the tagging stage. An additional second or third pass through the data would greatly reduce the effect of the read order in the tagging stage.

Yet another aspect of the pipeline that can be addressed in the future is the issue of alternative splicing, namely the presence of transcript isoforms with missing exons. If these transcripts are present in the input file, Kollector can still recruit genomic reads from their missing exons, as it does with intronic regions that are missing from the bait sequence in the first place. However, we already established that Kollector's success rate in assembling targets is adversely affected as the length of the missing sequence gets longer, so it can be argued that assembling these isoforms is more challenging. Another issue with using shorter isoforms is that

they can affect the reporting phase. For instance, a transcript with an alternative last exon can align perfectly to a gene assembly that lacks the terminal exon, which will lead to its misreporting as a successfully assembled target. For these reasons, it is advisable to use the full-length, or at least the longest isoform for each transcript input. However, this can be difficult to determine for users. In the future, Kollector can have a preprocessing step, where input transcripts are mapped to each other (preferably using a Bloom filter) and those that are completely subsumed by another transcript can be removed before the tagging stage.

In conclusion, we expect Kollector to be a valuable addition to the current suite of targeted assembly tools. Kollector is unique in taking full advantage of transciptome data. The Progressive Bloom filter is a novel take on the data structure, and it ensures that Kollector does scale to large datasets. Kollector can also work with incomplete and divergent input sequences, so it can be used to reconstruct orthologous sequences in non-model organisms. We fully expect that Kollector will find utility in the reconstruction of large gene regions *de novo*, which can be used to improve on draft genome assemblies, as we have demonstrated.

## Bibliography


Alkan, C., Coe, B.P. and Eichler, E.E. Genome structural variation discovery and genotyping. *Nat Rev Genet* 2011;12(5):363-376.

Alkan, C., Sajjadian, S. and Eichler, E.E. Limitations of next-generation genome sequence assembly. *Nat Meth* 2011;8(1):61-65.

Allen, J.M.*, et al.* aTRAM - automated target restricted assembly method: a fast method for assembling loci across divergent taxa from next-generation sequencing data. *BMC bioinformatics* 2015;16:98.

Altschul, S. Basic Local Alignment Search Tool. *Journal of Molecular Biology* 1990;215(3):403-410.

Baker, M. De novo genome assembly: what every biologist should know. *Nat Meth* 2012;9(4):333-337.

Bentley, D.R.*, et al.* Accurate whole human genome sequencing using reversible terminator chemistry. *Nature* 2008;456(7218):53-59.

Birney, E. Assemblies: the good, the bad, the ugly. *Nat Meth* 2011;8(1):59-60.

Bloom, B.H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* 1970;13(7):422-426.

Brankovics, B.*, et al.* GRAbB: Selective Assembly of Genomic Regions, a New Niche for Genomic Research. *PLoS computational biology* 2016;12(6):e1004753.

Brown, S.D.*, et al.* Neo-antigens predicted by tumor genome meta-analysis correlate with increased patient survival. *Genome Research* 2014;24(5):743-750.

Cancer Genome Atlas, N. Comprehensive genomic characterization of head and neck squamous cell carcinomas. *Nature* 2015;517(7536):576-582.

Chaisson, M.J., Wilson, R.K. and Eichler, E.E. Genetic variation and the de novo assembly of human genomes. *Nat Rev Genet* 2015;16(11):627-640.

Chevreux, B., Wetter, T. and Suhai, S. Genome Sequence Assembly Using Trace Signals and Additional Sequence Information. In, *Computer Science and Biology: Proceedings of the German Conference on Bioinformatics (GCB) 99*. 1999. p. 45-46.

Chu, J.*, et al.* BioBloom tools: fast, accurate and memory-efficient host species sequence screening using bloom filters. *Bioinformatics* 2014;30(23):3402-3404.

Edgar, R.C. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC bioinformatics* 2004;5:113.

Gish, W. and States, D.J. Identification of protein coding regions by database similarity search. *Nature genetics* 1993;3(3):266-272.

Goodwin, S., McPherson, J.D. and McCombie, W.R. Coming of age: ten years of next-generation sequencing technologies. *Nat Rev Genet* 2016;17(6):333-351.

Grabherr, M.G.*, et al.* Full-length transcriptome assembly from RNA-Seq data without a reference genome. *Nat Biotechnol* 2011;29(7):644-652.

Hammond, S.A.*, et al.* It's okay to be green: Draft genome of the North American bullfrog (Rana [Lithobates] catesbeiana). *bioRxiv* 2017.

Heather, J.M. and Chain, B. The sequence of sequencers: The history of sequencing DNA. *Genomics* 2016;107(1):1-8.
Hernandez, D.*, et al.* De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. *Genome Res* 2008;18(5):802-809.
Johnson, K.P.*, et al.* Rates of genomic divergence in humans, chimpanzees and their lice. *Proceedings of the Royal Society B: Biological Sciences* 2014;281(1777):20132174-20132174.
Langmead, B. and Salzberg, S.L. Fast gapped-read alignment with Bowtie 2. *Nat Methods* 2012;9(4):357-359.
Li, H. and Durbin, R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics* 2009;25(14):1754-1760.
Li, R.*, et al.* De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res* 2010;20(2):265-272.
Li, Z.*, et al.* Comparison of the two major classes of assembly algorithms: overlap-layout-consensus and de-bruijn-graph. *Brief Funct Genomics* 2012;11(1):25-37.
Moreton, J., Izquierdo, A. and Emes, R.D. Assembly, Assessment, and Availability of De novo Generated Eukaryotic Transcriptomes. *Frontiers in genetics* 2015;6:361.
Myers, E.W.*, et al.* A whole-genome assembly of Drosophila. *Science* 2000;287(5461):2196-2204.
Nagarajan, N. and Pop, M. Sequence assembly demystified. *Nat Rev Genet* 2013;14(3):157-167.
Parra, G., Bradnam, K. and Korf, I. CEGMA: a pipeline to accurately annotate core genes in eukaryotic genomes. *Bioinformatics* 2007;23(9):1061-1067.
Peng, Y.*, et al.* IDBA-tran: a more robust de novo de Bruijn graph assembler for transcriptomes with uneven expression levels. *Bioinformatics* 2013;29(13):i326-i334.
Peterlongo, P. and Chikhi, R. Mapsembler, targeted and micro assembly of large NGS datasets on a desktop computer. *BMC bioinformatics* 2012;13(1):48.
Pevzner, P.A., Tang, H. and Waterman, M.S. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences of the United States of America* 2001;98(17):9748-9753.
Pevzner, P.A., Tang, H. and Waterman, M.S. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences* 2001;98(17):9748-9753.
Reed, D.L.*, et al.* Genetic analysis of lice supports direct contact between modern and archaic humans. *PLoS biology* 2004;2(11):e340.
Robertson, G.*, et al.* De novo assembly and analysis of RNA-seq data. *Nature Methods* 2010;7(11):909-912.
Sakharkar, M.K., Chow, V.T. and Kangueane, P. Distributions of exons and introns in the human genome. *In silico biology* 2004;4(4):387-393.
Sanger, F., Nicklen, S. and Coulson, A.R. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America* 1977;74(12):5463-5467.
Simpson, J.T.*, et al.* ABySS: A parallel assembler for short read sequence data. *Genome Research* 2009;19(6):1117-1123.
Slater, G.S. and Birney, E. Automated generation of heuristics for biological sequence comparison. *BMC bioinformatics* 2005;6:31.
Stranneheim, H.*, et al.* Classification of DNA sequences using Bloom filters. *Bioinformatics* 2010;26(13):1595-1600.

Venn, O.*, et al.* Nonhuman genetics. Strong male bias drives germline mutation in chimpanzees. *Science* 2014;344(6189):1272-1275.
Voelkerding, K.V., Dames, S.A. and Durtschi, J.D. Next-generation sequencing: from basic research to diagnostics. *Clin Chem* 2009;55(4):641-658.
Wang, Z., Gerstein, M. and Snyder, M. RNA-Seq: a revolutionary tool for transcriptomics. *Nat Rev Genet* 2009;10(1):57-63.
Warren, R.L.*, et al.* Derivation of HLA types from shotgun sequence datasets. *Genome Medicine* 2012;4(12):95.
Warren, R.L. and Holt, R.A. Targeted Assembly of Short Sequence Reads. *PloS one* 2011;6(5):e19816.
Warren, R.L.*, et al.* Improved white spruce (Picea glauca) genome assemblies and annotation of large gene families of conifer terpenoid and phenolic defense metabolism. *The Plant journal : for cell and molecular biology* 2015;83(2):189-212.
Wu, T.D. and Watanabe, C.K. GMAP: a genomic mapping and alignment program for mRNA and EST sequences. *Bioinformatics* 2005;21(9):1859-1875.
Zerbino, D.R. and Birney, E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res* 2008;18(5):821-829.