Sackler Faculty of Exact Sciences,
Blavatnik School of Computer Science

On reducing complexity of deep sequencing data analysis

THESIS SUBMITTED FOR THE DEGREE OF
"DOCTOR OF PHILOSOPHY"
by

Roye Rozov

The work on this thesis has been carried out
under the joint supervision of
Prof. Ron Shamir
and
Prof. Eran Halperin

Submitted to the Senate of Tel-Aviv University
November 2017

Acknowledgments

This thesis summarizes my work over the last five years. During this period, I learned a new field, began a new career, and grew as a person. Naturally, these changes did not always proceed smoothly - such is life. In retrospect, it is clear to me it would not have been possible for me to complete the work in this thesis without a great deal of help from many generous people: I am indebted to them all.

I am extremely grateful to have had the opportunity to learn from and work with my wonderful advisers - Ron Shamir and Eran Halperin. Their endless patience proved crucial when I was 'learning the ropes' in the first stage of my PhD. Later, I learned to appreciate the breadth and depth of their guidance regarding research and life. They set a high bar to teach me how to leap.

My collaborators and lab mates enriched my knowledge and made the experience of doing research fun. I had the good fortune of being introduced into the world of plasmids and de novo assembly thanks to Itzik Mizrahi and his group. They showed me how challenging and rewarding it can be to produce a tool that is actually useful to biologists. I also had the great pleasure of working side by side with Gil Goldshlager. Our Summer spent working together was the most fun period I've had exchanging ideas at a white-board. My many lab mates in Ron and Eran's groups made hours spent working in the labs feel like time spent at home. I would also like to thank Gilit Zohar-Oren for always going out of her way to help all of us in any way possible.

I would like to thank the Edmond J. Safra Foundation for support throughout most of my PhD. I would also like to thank the Cloud Storage group at IBM Research, Tel Aviv, headed by Dalit Naor and Ronen Kat for hosting me and helping me win a PhD Fellowship to support part of my studies. I greatly enjoyed working with Ety Khaitzin and Danny Harnik during the period I was there.

At times, the conviction that there is nothing I would rather be doing and no other place I would rather be doing it was essential. This conviction stemmed from conversations in my youth with my brother Yadin, who instilled in me a deep love for science and Israel. Along with my brother, I am ever grateful for the constant love and support of my family.

My wife and love Shira was solely responsible for my spiritual wellbeing ever since we met. She always put my needs and those of others ahead of her own and showed me love unlike any I had ever known. She also brought our daughter Sol, a new level of joy, to this world. I owe her everything there is to give.

Preface

This thesis is based on the following three articles that were published throughout the PhD period in scientific journals:

1. R. Rozov, R.Shamir, E. Halperin; Fast lossless compression via cascading Bloom filters. BMC Bioinformatics 2014, 15 (Suppl 9):S7.

2. R. Rozov, A.B. Kav, D. Bogumil, N. Shterzer, E. Halperin, I. Mizrahi, R. Shamir; Recycler: an algorithm for detecting plasmids from de novo assembly graphs. Bioinformatics 2017; 33 (4): 475-482.

3. R. Rozov, G. Goldshlager, R.Shamir, E. Halperin; Faucet: streaming de novo assembly graph construction. To appear, Bioinformatics, 2017

Abstract

The volume of deep sequencing data generated over the last decade has grown exponentially, as machines that output many millions of short sequences in parallel replaced slower Sanger-sequencing machines that read individual DNA molecules. As a result of this change, throughput has exploded and the price per base has plummeted by five orders or magnitude. This data growth means that the time required to perform many standard processing tasks, such as alignment and assembly, also grows exponentially if algorithms that were designed for much smaller numbers (or lower volumes) of sequences are applied. In this thesis, we introduce new techniques designed to address these challenges. We use advanced data structures to compress and index sequence data, aiming to allow efficient querying with reduced storage space. In the context of sequence compression, we demonstrate a faster approach than extant methods to compress read sequences relative to a reference by representing sequences using Bloom filters. In the context of de novo assembly, we first develop a new approach to assembling circular sequences using assembly graphs as an index. Then, we demonstrate a streaming approach to assembly graph construction to reduce construction time, memory, and disk use. We also show this construction enables most salient information present in the read data to be maintained, leading to highly contiguous assemblies.

Contents

Chapter 1

Introduction

Mankind's ability to read DNA improved dramatically in the first decade of the twenty-first century. The huge costs required to complete sequencing and assembly of the first Human genome via Sanger sequencing spurred development of different technologies to speed up sequencing, make it cheaper, and increase throughput. Several companies introduced competing technologies to address these needs. Collectively, these technologies are commonly referred to as "Deep," or previously, "Next Generation" Sequencing technologies. One of those companies now controls 70% of the sequencing market - Ilumina [1] - by virtue of their Sequencing by Synthesis (SBS) technology. This technology was obtained via acquisition of a smaller company called Solexa in 2007 [2] that had marketed the first sequencer employing SBS a year earlier.

Development of deep sequencing over the past decade led to an over five order of magnitude drop in price, and seven order of magnitude increase in throughput relative to Sanger sequencing [3] [Figure 1.1]. It now costs about $1000 to sequence a human genome, whereas in 2001 it cost over $200 M. This price drop has had profound effects on biological research. For example, It is now possible to perform studies of hundreds of thousands of individuals to finely characterize variation in humans [4]. Also, nearly a hundred new assays have emerged that are based on sequencing [5]. In some cases, as in mRNA sequencing used to measure transcription (called RNA-Seq), sequencing has largely supplanted existing technology (in this case, microarray chips) due to the much higher resolution enabled; in others, such as Chromosome Conformation Capture (3C) [6] for the purpose of assaying 3D

genome structure, sequencing based assays are the first high-throughput means of investigation available.

Along with development of new applications of deep sequencing, new technologies have emerged that aim to address its shortcomings. Third generation technologies generate reads that are tens to thousands of times longer than SBS reads [7], and are sequenced individually, and thus not prone to biases introduced by PCR amplification. Currently, such reads have a much higher error rate than SBS reads and are much more expensive per base, but they have already been shown to provide tremendous benefits for de novo assembly [8], and in differentiating between closely related transcripts in RNA-Seq [9, 10], or closely related strains of viruses in metagenome sequencing [11, 12].

This rapidly shifting technology landscape has created tremendous new bioinformatic challenges. The large number of sequences per sample, coupled with the ever-increasing number of studies and size per study enabled by cheap sequencing, have led to a deluge of data. To scale analysis tasks, algorithms have evolved along with sequencing technologies. Fields thought to be mature such as sequence alignment, where optimal algorithms are known for aligning individual sequences [13, 14], and good heuristic algorithms exist for aligning individual sequences to large databases [15], have required a steady stream of innovations to cope with new challenges such as aligning millions of very short sequences to large mammalian genomes [16, 17], searching for relevant datasets among the thousands that are publicly available [18, 19, 20] (where each is composed of millions of sequences), and efficiently approximating all-versus-all pairwise comparisons of millions of reads in order to perform de novo assembly [21, 8, 22]. Cheap sequencing also enabled an endeavor to assemble more genomes, larger genomes, and collections of species via metagenome sequencing or transcripts via RNA-Seq. The challenges in this area demanded ever larger space to represent large collections of sequences simultaneously. In both cases, new techniques to index or reorder sequences proved to be essential.

Such techniques are the focus of this thesis. We begin with a detailed description of sequencing technologies and the magnitude of their data production. Then, we characterize the challenge these data impose on traditional algorithms on strings. Finally, we discuss modern string indexing approaches before describing our own contributions that are based on them. We conclude by providing some perspective regarding future developments of sequencing technologies and the tools that will be

needed to analyze their output.

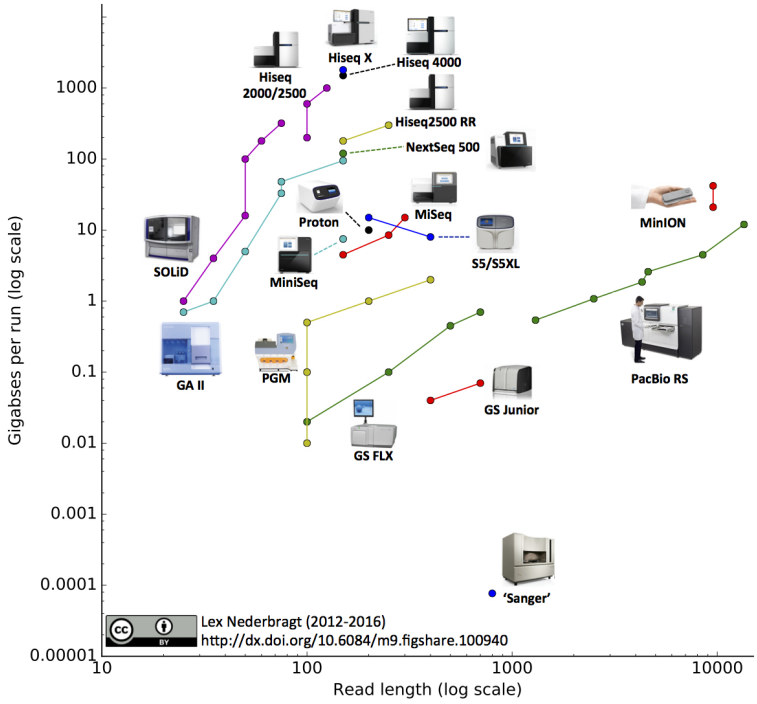

Figure 1.1: Development of sequencing technologies. Read length is plotted against throughput in Gigabases per run for various contemporary and extinct sequencing technologies. Source: [3]

1.1 Deep sequencing

Deep sequencing involves sequencing large numbers of short DNA molecules in parallel. The most popular approach is Illumina's Sequencing By Synthesis (SBS), involving fragmenting DNA to a desired size distribution, ligating adapters to fragment ends, fixing DNA ends to a glass slide, and recording fluorescence emitted as new fluorescently tagged nucleotides in solution are added to bound molecules [Figure 1.2]. Modified nucleotides called reversible terminators are introduced to the solution along with DNA polymerase, and the action of DNA polymerase is used to add nucleotides to fixed single-stranded templates. *Reversible terminators* are

nucleotides that are modified to stop extension when incorporated, and stop their termination activity along with emitting a nucleotide-specific color when struck with light; this allows for signaling a particular base's incorporation and recovering this information via imaging. This process is repeated for up to 300 cycles, a limit imposed by image signal degradation limiting resolution at greater lengths.

The input DNA is referred to as the sequenced sample. Stacking of images generated during the course of sequencing and tracking bases incorporated sequentially at fixed locations allows reconstruction of millions of sequences in parallel. Sequencing errors are primarily substitutions, on the order of one out of every thousand bases, and insertions or deletions occur on the order of one out of every million bases [24]. Individual sequences are referred to as *reads*. One approach to circumvent the length limitation is to generate *paired-end reads*, where instead of sequencing up to the maximal length in one direction, opposite ends of a fragment are sequenced. These pairs yield information about reads that originate from the same molecule at an approximately known distance; such knowledge is often informative in aligning reads to unique positions on a reference genome or for resolving repeats in assembly. The sequence between these pairs remains unknown, but the distribution of distances between them can be estimated by mapping to a reference or to assembled contigs. For each such pair, the *insert size* refers to the size of the source molecule, i.e. the inferred distance between individual mates of the pair (gap size) plus the lengths of the read sequences themselves. Another important parameter for sequencing applications is *coverage*, which can mean either the proportion of the source material (genome, or collection of genomes in the case of microbiomes) that has been successfully sequenced, or the average number of reads covering a base in the source material. In this thesis, coverage refers to the latter unless otherwise specified.

1.2 Third generation sequencing

Recently, new technologies have emerged to overcome some of the limitations of SBS [25]. These methods provide either longer-range information or aim to remove systematic bias of SBS due to GC content [26]. Collectively, these are referred to as Third Generation sequencing technologies.

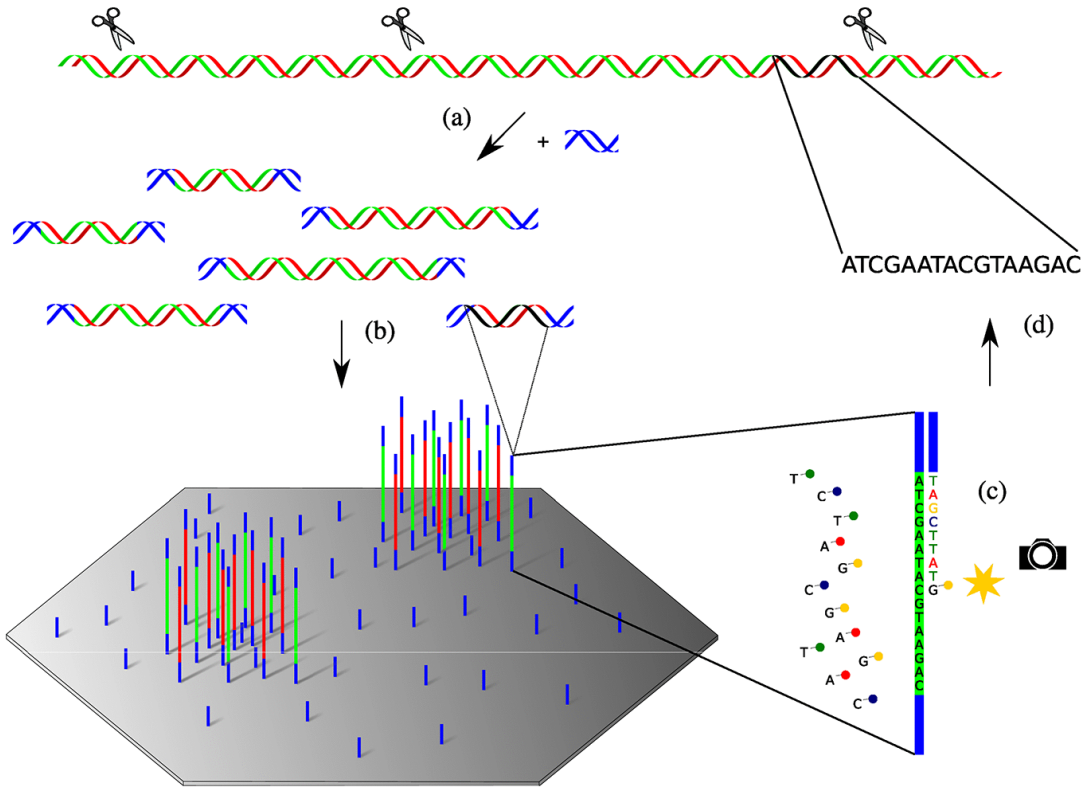

Figure 1.2: An illustration of sequencing by synthesis (SBS). a) input DNA molecules are fragmented to a desired size distribution. b) adapters are ligated to fragment ends. These fragments are then placed on a glass slide coated with a lawn of adapters complementary to those placed on input fragments, causing them to adhere to the surface. These adhered molecules are then amplified and processed to leave only bound single ends. c) reversible terminators are introduced to the solution. These are harnessed by polymerase to extend complementary strands of bound fragments but lead synthesis to stop after each base extension. Illumination leads their terminator function to cease and a fluorescent colored signal corresponding to the incorporated base's identity is emitted. d) recording successive emissions at each position of the imaged slide allows construction of the sequence present there. The sequence from each position is referred to as a 'read' Source: [23]

1.2.1 Single molecule technologies

Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT) both manufacture sequencers that can generate reads tens of thousands of base-pairs long. PacBio's Single Molecule Real Time (SMRT) technology is based on imaging of individual bound polymerase proteins controling the advance of large single DNA molecules inside very small wells called Zero Mode Waveguides (ZMWs). These wells are designed to be small enough to prevent wavelengths of light from efficiently passing, thus causing attenuation and increasing imaging resolution at the base of the well. This allows reading of minute flouresence signals emitted as a result of base incorporations by the action of *individual* polymerase molecules [27]. Thousands of wells are sequenced in parallel. SMRT sequencing avoids the need for amplification used to boost signal strength in SBS and thus reduces bias due to sequence content, but PacBio sequencing is prone to a much higher rate of indels than SBS, and shows a raw error rate of 10%. High consensus accuracy can be achieved with high coverage [28, 29].

ONT's nanopore sequencers measure current changes resulting from the electrophoresis induced movement of DNA nucleotides through a transmembrane protein. Nanopore sequencing requires no optical measurement or chemical labeling of DNA, allowing for a very small instrument size. ONT's MinIon is about the size of an office stapler. Competing methods require instruments with sizes comparable to small printers up to large refrigerators. This size, combined with the ability to sequence in real time have made MinIons popular for field sequencing applications, allowing for on site monitoring [12], and tracking of virus spread and evolution [11].

Although both PacBio and ONT instruments sequence single molecules at roughly the same error rate, sequence characteristics of ONT reads differ from PacBio's in several ways. There have been recent reports that reads can span hundreds of kbp using careful DNA library preparation [7], [30]. A systematic assessment of ONT reads showed some GC bias effects [31], and consensus accuracy shown in a recent ONT human de novo assembly project [30] looks to be effected as a result. The MinIon is also capable of reading DNA modifications, as shown in a recent report highlighting methylation sequencing [32].

Both technologies are currently much more expensive on a per-nucleotide basis than SBS reads, leading their use to be limited to sequencing small genomes or to

supplementing higher coverage SBS reads. Current throughput available from these companies' sequencers is lower than that which can be obtained by Illumina's HiSeq intstruments, but ONT claims their PromethIon instrument will produce multiple Tb per day, matching or exceeding HiSeq outputs, and PacBio is also expected to introduce throughput upgrades in the near future [33].

1.2.2 Synthetic long reads

An alternative approach to obtaining long range information without expensive single molecule sequencing technologies is modifying SBS library prep to retain long range information present in the source DNA. One approach to obtaining long range information is by generating large (up to tens of kb) fragment libraries, splitting individual fragments into many pools, and appending pool-specific bar-codes before applying SBS [25]. Reads generated from this process are termed 'synthetic long reads' and convey long range information in that individual bar-codes correspond to common source molecules. Such reads are marketed by Illumina and 10X Genomics. As both rely on SBS, they retain the high per-base accuracy of Illumina reads, but also inherit their GC bias. Thus, assemblies generated with 10X reads achieve highly contiguous scaffolds, but retain many more gaps than contigs obtained with single molecule technologies.

1.3 Growth of sequencing data generated

A recent study assessed the state of sequencing data production in depth [34]. We summarize and quote the following facts from that study, but as this assessment was made two years ago, they serve as lower bounds. Over the past decade, sequence data generated has doubled roughly every seven months. The estimated current worldwide sequencing capacity exceeds 35 petabases per year, while more than 100 petabytes of storage are currently used by only 20 of the largest institutions performing sequencing. If the growth continues at the current rate, it is expected that one exabase of sequence will be generated per year in the next five years and approach one zettabase of sequence per year by 2025 [Figure 1.3]. More conservative estimates of doubling every 12 - 18 months (equivalent to Moores law), imply exabase-scale aggregate data will be reached within the next decade. This makes genomics one

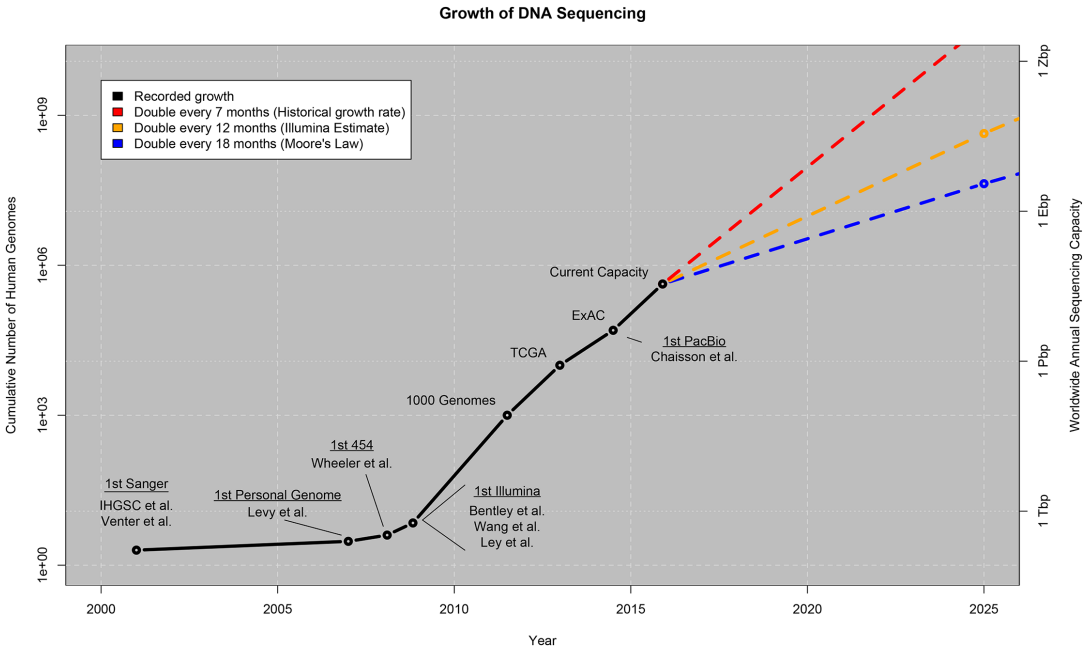

Figure 1.3: Past and projected DNA sequence data generation. Rapid growth in throughput and declines in price-per base sequenced led to the emergence of large scale proejcts and greatly increased aggregate sequence output. The three colored dashed curves portray future aggregated output at several possible growth rates. Source: [34]

of the fields generating the most data worldwide, comparable with much-discussed websites such as Youtube and Twitter, and traditional data-intensive fields such as Astrophysics [34].

As throughput grew, the cost of sequencing each base dropped precipitously. Whereas the first human genome was sequenced over 13 years at a cost of $300 million - $2.7 Billion (depending on what costs are included) [35], the highest throughput Illumina machine, called the X10, allows for generation of about 1 Tb per day, or roughly 10 human genomes at 30-fold coverage at a cost of about $1000 per genome. This drop in cost has opened the door to myriad experimental assays that were previously inaccessible: population-level sequencing of hundreds of thousands of genomes from different countries [4], sequencing of thousands of individual cells from all tissues of the human body to create a Human Cell Atlas [**?**] etc.

Such exponential growth in aggregate data, and in datasets emerging from such large scale studies, presents new challenges. Sequencing centers must contend with

very high current outputs, and with rapid increases in output rates. This makes anticipation of storage costs difficult for budgeting. In addition, whereas in other fields, raw data is often discarded as it is produced, the gradual maturation of analysis methods leading to higher accuracy and sensitivity in results has led to reluctance of genomics researchers to discard raw read data once downstream results have been produced. Keeping raw data allows for the opportunity to revisit and reanalyze raw data as needed. The introduction of cloud storage and computational resources to genomics makes scaling storage capacity simpler; however, high costs for storage persist.

Making data manageable and analysis on such data feasible have repeatedly necessitated development of entirely new approaches. As noted by Loh et al [36], "Any computational analysis, such as sequence search, that runs on the full genomic library - or even a constant fraction thereof - scales at least linearly in time with respect to the size of the library and therefore effectively grows exponentially slower every year." While sequencing centers mainly contend with storage and management, individual labs must address the need to analyze much larger datasets than were available or possible before. Contemporary researchers often wish to analyze many samples together to find rare variants or strains, or to achieve sufficient statistical power to be able to differentiate between groups. Also, sequencing has become cheap enough that it is now possible to probe DNA samples derived from large polyploid plant genomes sequenced to high coverage, or metagenomes from diverse environments. In either case, it becomes necessary to store, search within, and summarize over extremely large data volumes.

1.4 Traditional text indexing approaches

Index structures have been applied to text processing for decades [37]. Full text indexes like suffix tries and arrays allow finding arbitrary length exact matches of patterns in a text. Hash-based indexes allow matching of fixed length patterns and therefore introduce a trade-off: sensitivity decreases while specificity increases as the pattern length grows. Although construction time and space complexity are linear, there are usually large constant costs associated in terms of the space required to build or store these structures in memory. While index construction is computationally expensive, usually the one-time cost of construction is offset by the

time efficiency granted by allowing repeated efficient querying of the text without scanning its entirety.

Full-text indexes effectively compress the text being indexed, in that when indexing a collection of strings, each substring will appear only once. However, as their main purpose is exact matching of patterns and not space efficient encoding, data-compression specific indexes provide more efficient reductions in data volume. For example, Huffman coding involves counting appearances of characters to minimize the number of bits used per character encoding of a given text [Huffman1952]. The Burrows-Wheeler transform reorders text in a manner that increases the tendency of identical characters to be adjacent, leading to better compression via run length encoding of characters [38]. Lempel-Ziv encoding builds a dictionary of previously seen substrings that are later referred back to to append gradually longer strings that may recur [39]. Also, in specialized scenarios such as when the order inputs appear can be discarded or the types of queries that will be made on the data can be anticipated, even more efficient, specialized data structures can be used. For example, Bloom filters [40] compactly represent a set and provide a simple mechanism for querying set membership. They trade space efficiency for the possibility of false positive results, and have proven very effective as caches meant to avoid costly lookups in storage [41].

Alignment of sequences also greatly benefits from indexing. When a pattern is searched for in a database of sequences, generating a hash of k-mer substrings present in the database mapping to sequences containing those k-mers is much more efficient than searching in every sequence in the database independently. Such seed based alignment greatly benefits efficiency, but may somewhat reduce sensitivity, e.g. in cases where the query shares no k-mers with the index but shares some when one mismatch is allowed. This type of indexing and tracking of k-mer positions in the database to allow hit extensions in dynamic programming is the idea behind the most popular bioinformatics tool to date, BLAST (Basic Local Alignment Search Tool) [15]. Such indexing is also used in alignment of genomes against each other in BLAT (BLAST-like alignment tool) [42] and MUMmer [43].

When de novo assembly of genomes is performed, an intermediate representation of input sequences called an *overlap graph* is first constructed [44]. Nodes in such graphs represent input sequences, and edges represent overlaps among them. Overlaps may be constant or of a range of lengths, and they may be exact matches

or alignments exceeding some significance threshold. As the input data to an assembler is a set of sequenced fragments that may contain errors, the purpose of summarizing these sequences in a graph is to summarize the set of all *possible* adjacencies between input sequences. As with full text indexes, these graphs also have the advantage of greatly reducing data representation size by encoding substrings repeated in the reads only once. The graph is later used to generate a walk corresponding to the source genome. Generating candidate walks and choosing one that is in best agreement with various characteristics of the input data are the main challenges of assembly. The two types of graphs in broad use are de Bruijn graphs (DBGs) and string graphs. Recently, more applications of these graphs have begun to emerge, such as graph-based variant calling, and concise queryable representation of variation at the population level [45, 46].

1.5 Indexing modern sequence data

Suffix tries, suffix arrays, and hash tables often require tens of GB of RAM to represent large (e.g. mammalian) genomes and hundreds of GB to represent large read collections, and thus become impractical as data sizes grow. Needleman-Wunsch [14] and Smith-Waterman [13] dynamic programming based alignment yield optimal results for global and local alignment, respectively, but are expensive in terms of time and memory for each sequence aligned. In the context of aligning SBS reads that are near exact matches to the reference genome, Smith-Waterman alignment is also wasteful in that very few base modifications need to be considered on average, making approaches that rely on an indexed reference and light modifications of exact-matching algorithms more appealing. Also, SBS usually involves sequencing to high coverage, introducing a large amount of redundancy among the reads. Generic compression approaches such as gzip converge to optimal compression rates as data volumes approach infinity, however they hold no such guarantees for finite data, and thus often benefit from sorting data or compression relative to a reference genome. Taken together, these failings of traditional, general-purposes approaches observed upon their application to SBS data motivate the need for tailor-made solutions that scale better.

1.5.1 Data structures and techniques for indexing, grouping, and summarizing reads

BWT, FM index of genomes and reads

The Burrows Wheeler Transform (BWT) [38] is a permutation of a text T allowing reconstruction of T. Its naive construction involves collecting all cyclic permutations of T into a matrix M, sorting these permutations lexicographically, and extracting the last column of the matrix formed by the sorted cyclic permutations. More efficient construction algorithms exist that avoid explicit storage of all the cyclic permutations [47]. It is a full text index, allowing matching of any substring in T via a backwards search mechanism. Its space use is nearly optimal, in that it requires no pointers for its representation - only an array of length nearly equal to the original text.

The FM (either abbreviating Full-text in Minute space, or its inventor's last names, Ferragina and Manzini) index [48] is a set of auxiliary data structures providing additional capabilities to the BWT. These data structures correspond to a the first column of the M described above, and a sampling (i.e. every k-th row for some constant k) of the suffix array of T. Taken together, patterns can be matched and their positions in T can be extracted in $O(P)$ where P is the pattern searched for.

Bloom filters

A Bloom filter [40] is a bit array B of size m initially set to all 0 values, paired with h hash functions $f_1, \ldots, f_h$ such that the range of each is $[1, m]$. To insert an element x to B means to apply each hash function in turn and set the position returned by each to 1 in B (bits previously set to 1 are left as 1). To query if an element x is in B similarly involves evaluating whether $f_i(x) = 1$ for all $i \in [1, h]$. If at least one function returns 0, then by definition x has not been inserted to B. However, if all hash functions return 0 it is probable but not certain that x was inserted to B, as some of the functions may return 1 due to hash collisions. The rate F such false positives are returned may be set by varying m and h relative to n, the number of elements to be inserted. Using optimal parameter choices [41], the space per element required is $1.44 \log_2(\frac{1}{F})$.

Bloom filters provide approximate set representations. They are probabilistic in that queries made to them may yield false positive answers, but not false negatives. The advantages they provide are constant space cost per element inserted (regardless of length) and querying time that is linear in the length of sequence inserted.

Read de Bruijn graphs

A de Bruijn graph (DBG), $G = (V, E)$, is a directed graph defined over an alphabet Σ as the set of nodes v, such that $v \in V \iff v \in \Sigma^k$ and arcs $(u, v) \in E \iff u[2:k] = v[1:k-1]$, where $x[i:j]$ denotes the substring of string x from position i to j, inclusive of the ends. In other words, a DBG is a directed graph having k-mer strings derived from Σ as nodes, and arcs representing $k-1$ character overlaps between k-mers. In the context of indexing reads, usually a DBG will refer to the subgraph of the formally defined graph over $\Sigma = \{A, C, G, T\}$ induced by the set of k-mers observed in the reads, for some chosen value of k.

DBGs are simple to construct and reason about, leading them to lay at the heart of many index-based algorithms. To save memory, the arcs of a DBG need not be stored explicitly - given a node v in the G, one may query for the presence of $v[2:k]$ concatenated as a prefix of each member of Σ to extract all neighbors in G. Thus, DBGs are naively represented as fixed length (4^k) array of k-mer counts when k is small (i.e., $k \leq 15$) enough to allow such an array to be held in memory, and as hash tables when k is larger. In the former, array positions correspond to the binary encoding of k-mers, and in the latter k-mers are keys and values are their counts. The essential common aspect of all DBG representations is the ability to query for the presence of extensions of a given k-mer known to be in the graph.

The transformation from read sequences to the set of k-mers in the reads is lossy, in that it is not possible to recover counts of consecutive repetitions of k-mers in the reads, leading to collapse of some sequences. Similarly, representing a DBG with only the set of k-mers and discarding arcs may lead to false arcs (u, v) when both u and v are in V but (u, v) is not observed on any read; to avoid such errors, arcs can be stored instead of nodes. Fortunately, any read sequence can be represented as a walk on G. Once a read's sequence has been mapped to the graph, it allows for enumeration of all possible extensions at the read's ends, and for comparisons with or mapping to previously seen sequences. The former enables DBGs to be used for de novo assembly, whereas the latter allows for DBGs to be used to find variants or

to represent collections of related samples, also referred to as pangenomes.

Binning reads by minimizers and MinHash

An important improvement over storage of all k-mers in a database as originally done for BLAST [15] is identification and storage of minimizers only. This concept was introduced independently in plagiarism detection [49] and in computational biology [50]. An (l, k) *minimizer* is the lexicographically minimal $l - mer$ inside a window of length $k \geq l$. Minimizers are useful in that a minimizer is shared by any pair of sequences having a k base overlap. Minimizers were originally intended to reduce the size of hash tables used for seed-and-extend based alignment: since length l minimizers essentially 'cover' k base windows and are sparse in number relative to them, they allow for efficient seeding while reducing storage requirements. Also, since sequences sharing minimizers tend to overlap, binning based on minimizers can be used to increase efficiency of all-pairs overlap needed to compare groups of sequences against each other or to align the entirety of a sequence database against itself. Any l-mer order can be used instead of lexicographic order to determine minimality.

Recently, an improvement over naive generation of minimizers was introduced called universal hitting sets [51]. A universal hitting set is a set S of k-mers guaranteed to have a member of S included in every sequence of length L, where $L > k$. This work showed that universal hitting sets can reduce the number of minimizers needed to bin groups of sequences and increase the average distance between them. An extension of this work highlighted the advantages of not using lexicographical ordering in determining minima [52].

A related approach, called min-wise locality sensitive hashing, or MinHash [53], also allows binning of sequences that are likely to be similar together. MinHash differs from minimizers in its use of multiple applications of random hashing in order to increase the probability of overlaps inside each generated bin. MinHash utilizes grouping based on an estimate of the Jaccard similarity measure $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$, the ratio of the size of intersection to the size of union of two sets A and B.

For each set $S = \{s_1 \dots s_n\}$ of interest, MinHash forms a fingerprint of j elements that is used for comparsion between sets. j hash functions $h_1 \dots h_j$ that permute elements are applied to every element of S and the fingerprint F is the set of min-

ima under each hash function $f_i = \min\{h_i(s_1) \dots h_i(s_n)\}$, $i \in [1, j]$. In [53], it is shown that the number of hash collisions (i.e., matching minima within fingerprints) between sets A and B divided by j is an unbiased estimator for $J(A, B)$. Multiple permutations are needed in order to reduce the variance of this estimator.

Streaming read data

Streaming algorithms were formally characterized in 1996 in the seminal work of Alon et al. [54]. The streaming model describes operations performed on continuously arriving data where a fixed amount of operations can be performed per data element with fixed memory. The stream can only be traversed a small number of times, or only once. This is meant to convey either data that is continuously being produced, or is too large for storage.

The benefits of the streaming approach are a consequence of the constraints it imposes. Streaming algorithms by definition use low memory and few operations per data element. Streaming algorithms may also serve to reduce storage space as it is possible to store only the results they generate instead of the data or index used to analyze it.

1.5.2 Applications demanding indexing

Error correction, k-mer counting and filtration

Counting k-mers is a basic preprocessing step employed as part of many sequence analysis tasks. Its most popular application is removal of rare error k-mers for the sake of de novo assembly [55]. This is motivated by the intuition that every mismatch in a read caused by a sequencing error will create up to k erroneous k-mers, and each of these will be much less frequent than real k-mers if error positions are uniformly sampled.

Since large sequencing experiments can lead to many billions of k-mers being generated, naive counting via hash tables requires very high memory. As a result, various approaches have been used to allow fast and low memory counting, including representation of k-mers by Bloom filters [56], partitioning to files on disk via minimizers [57, 58], and most recently, lossy counting and streaming filtration of reads and k-mers [59, 60, 61].

Mapping to a reference genome or collection

With the introduction of the short read alignment tools Bowtie and BWA in 2009 [17, 16], it was shown that the full-text searching capability of suffix arrays could be achieved using little space by indexing reference genomes using the BWT and extracting reference positions using the complementary FM-index. These tools greatly improved the state of the art in mapping, reducing run-times by several orders of magnitude versus earlier hash-based methods. More recent methods have shown how compression of reference databases or read sets against themselves so as to reduce redundancy within each can greatly improve speed of alignment with little loss in sensitivity. These methods have demonstrated such compressive approaches benefit both nucleotide or protein level dynamic programming based alignment as done by BLAST [36], and in near-exact match based mapping of short reads as done in Bowtie or BWA [62].

In the context of mapping for the sake of quantification, it has been shown that probabilistic assignment to reference sequences (typically transcripts or microbes in a metagenome) does not require full alignment, and can be made much faster by replacing alignment with simple assignment to groups of sequences that are compatible with the fragment that is to be assigned. These groups are called equivalence classes - a concept introduced in [63]. Recently, DBG based indexes of reference sequences have been used for fast assignment to equivalence classes. This mechanism forms the heart of the fastest transcript abundance estimation algorithms available, Kallisto [64] and Salmon [65]. This mechanism has also been used for quantification of species in metagenomes [66].

Compression of short reads

Compression draws on efficient indexing to allow matching among repeated substrings, and encoding meant to minimize the space consumed per character. Short read compression tools either compress reads relative to some reference or use reference-free approaches that typically involve reorganizing (e.g. sorting or binning) reads to increase the tendency of similar substrings to be near each other. Generally, reference based methods achieve lower compression ratios (defined as the ratio of output to input file sizes) while reference-free methods are faster [67].

The reason for this dichotomy is that the computational cost of mapping to a ref-

erence provides a clear benefit: if the reference closely matches the read's sequence, the read's sequence need not be stored: the position on the reference having that sequence can be stored instead. The time to align accounts for most of the time difference. If differences exist they are then written out along with the reference position, and then they can be encoded efficiently according to their distribution (e.g. by Huffman coding). This is the basis for several reference based compression tools (e.g. SlimGene [68], DeeZ [69]).

To reorganize reads, various approaches have been employed. The SCALCE algorithm [70] sorts reads according to presence of common substrings, with the intent of optimizing compression by a fixed size buffer, as done by gzip [39]. More recent methods have performed similar reorganization based on binning according to the presence of minimizers [71, 72]. In Chapter 2 of this thesis, we present a hybrid approach that avoids mapping to a reference, instead performing exact matching via Bloom filter hashing and compression of non-matching reads via SCALCE.

Some reference free methods also compress reads as paths on the DBG of the reads. In that case the graph must be stored and compressed along with the reads in order to allow for decompression. To this end, Bloom filter and other succint data structure graph representations have been employed [73, 74]. The more recent QUARK [75] uses a DBG for quasimapping and encoding, but does not require the reference set of transcripts for the sake of decompression.

De novo assembly of short reads

When performing de novo assembly, DBGs with large values of k are often used, and for large samples, tens to hundreds of billions of k-mers may need to be represented. In these situations, using hash tables to represent DBGs may require hundreds of GB of RAM due to the large number of keys and hash table implementation overhead. To reduce the need for such hardware requirements, Bloom filters have been used as an alternative means of representing DBGs [76, 77, 78].

Beyond the use of Bloom filters, more recent innovations include the use of minimizers to reduce memory in graph compaction and better parallelized processing [79, 80]. The efficiency of k-mer Bloom filters, often having dependency between consecutive queries, was improved in [81] and later this method was used for efficient weighted (where nodes are labeled with coverage) DBG construction [82]. Finally,

in Chapter 4 of this thesis, and earlier in the TwoPaCo algorithm for efficient DBG compaction [83], a representation combining identification of the set of junctions in the DBG along with a BF storing nodes of the DBG (to allow for traversal between junctions) was used.

De novo assembly of long reads

Currently, the most popular means of assembling long reads is based on the Overlap-Layout-Consensus (OLC) approach. This approach begins with finding all pairwise overlaps between reads. As such, initial assemblies of large genomes took thousands of hours of CPU time [8] [22]. A major advance past this computational hurdle was the change from performing all-pairs overlap assessment to first applying MinHash on all reads to bin them, and then confining alignments among pairs to be performed only among reads that are sufficiently similar to be in the same bin [8]. More recent approaches have similarly used minimizers and sorting to achieve even greater speed [84, 85], even without the requirement of initial polishing (error removal) of the error-prone reads before assembly.

Searching for relevant experiments

A new application enabled by the explosive growth of sequencing is search for relevant datasets among all publicly available samples. The goal of such search is to find samples likely to contain sequences that are similar to some query (or query set) of interest. The first work to propose this application by Solomon et al. [18] employed a binary tree of Bloom filter nodes, where each parent node was the bit-wise union of its children's bit arrays, and leaf nodes corresponded to Bloom filters of sequencing experiment k-mers. This work offered a dramatically more scalable solution than aligning queries of interest against every read data set individually by limiting explicit alignments to much smaller subtrees. Recent follow-up works have since improved memory efficiency and run times specifically when queries are sets [86, 87].

An alternative approach using minHash was also recently introduced. In this case, fingerprints are generated for each dataset, and a distance measure based on Jaccard similarity is used to select datasets matching a query set very quickly. This scheme enabled fast clustering by sequence similarity of all RefSeq genomes

in minutes [20] and fast identification of relevant samples out of all of the SRA database [19].

1.6 Summary of articles included in this thesis

Fast lossless compression via cascading Bloom filters R. Rozov, R.Shamir, E. Halperin; BMC Bioinformatics 2014, 15 (Suppl 9):S7.

Background: Data from large Next Generation Sequencing (NGS) experiments present challenges both in terms of costs associated with storage and in time required for file transfer. It is sometimes possible to store only a summary relevant to particular applications, but generally it is desirable to keep all information needed to revisit experimental results in the future. Thus, the need for efficient lossless compression methods for NGS reads arises. It has been shown that NGS-specific compression schemes can improve results over generic compression methods, such as the Lempel-Ziv algorithm, Burrows-Wheeler transform, or Arithmetic Coding. When a reference genome is available, effective compression can be ac hieved by first aligning the reads to the reference genome, and then encoding each read using the alignment position combined with the differences in the read relative to the reference. These reference-based methods have been shown to compress better than reference-free schemes, but the alignment step they require demands several hours of CPU time on a typical dataset, whereas reference-free methods can usually compress in minutes.

Results: We present a new approach that achieves highly efficient compression by using a reference genome, but completely circumvents the need for alignment, affording a great reduction in the time needed to compress. In contrast to reference-based methods that first align reads to the genome, we hash all reads into Bloom filters to encode, and decode by querying the same Bloom filters using read-length subsequences of the reference genome. Further compression is achieved by using a cascade of such filters.

Conclusions: Our method, called BARCODE, runs an order of magnitude faster than reference-based methods, while compressing an order of magnitude better than reference-free methods, over a broad range of sequencing coverage. In high coverage (50-100 fold), compared to the best tested compressors, BARCODE saves 80-90% of the running time while only increasing space slightly.

Recycler: an algorithm for detecting plasmids from de novo assembly graphs R. Rozov, A.B. Kav, D. Bogumil, N. Shterzer, E. Halperin, I. Mizrahi, R. Shamir; Bioinformatics 2017; 33 (4): 475-482.

Motivation: Plasmids and other mobile elements are central contributors to microbial evolution and genome innovation. Recently, they have been found to have important roles in antibiotic resistance and in affecting production of metabolites used in industrial and agricultural applications. However, their characterization through deep sequencing remains challenging, in spite of rapid drops in cost and throughput increases for sequencing. Here, we attempt to ameliorate this situation by introducing a new circular element assembly algorithm, leveraging assembly graphs provided by a conventionalde novoassembler and alignments of paired-end reads to assemble cyclic sequences likely to be plasmids, phages and other circular elements.

Results: We introduce Recycler, the first tool that can extract complete circular contigs from sequence data of isolate microbial genomes, plasmidome and metagenome sequence data. We show that Recycler greatly increases the number of true plasmids recovered relative to other approaches while remaining highly accurate. We demonstrate this trend via simulations of plasmidomes, comparisons of predictions with reference data for isolate samples, and assessments of annotation accuracy on metagenome data. In addition, we provide validation by DNA amplification of 77 plasmids predicted by Recycler from the different sequenced samples in which Recycler showed mean accuracy of 89% across all data types - isolate, microbiome and plasmidome.

Faucet: streaming de novo assembly graph construction R. Rozov, G. Goldshlager, R.Shamir, E. Halperin; Bioinformatics, btx471, 2017

Motivation: We present Faucet, a 2-pass streaming algorithm for assembly graph construction. Faucet builds an assembly graph incrementally as each read is processed. Thus, reads need not be stored locally, as they can be processed while downloading data and then discarded. We demonstrate this functionality by performing streaming graph assembly of publicly available data, and observe that the ratio of disk use to raw data size decreases as coverage is increased.

Results: Faucet pairs the de Bruijn graph obtained from the reads with additional meta-data derived from them. We show these metadata - coverage counts

collected at junction k-mers and connections bridging between junction pairs - contain most salient information needed for assembly, and demonstrate they enable cleaning of metagenome assembly graphs, greatly improving contiguity while maintaining accuracy. We compared Faucet's resource use and assembly quality to state of the art metagenome assemblers, as well as leading resource-efficient genome assemblers. Faucet used orders of magnitude less time and disk space than the specialized metagenome assemblers MetaSPAdes and Megahit, while also improving on their memory use; this broadly matched performance of other assemblers optimizing resource efficiency - namely, Minia and LightAssembler. However, on metagenomes tested, Faucets outputs had 14-110% higher mean NGA50 lengths compared to Minia, and 2-11-fold higher mean NGA50 lengths compared to LightAssembler, the only other streaming assembler available.

Chapter 2

Fast lossless compression via cascading Bloom filters

Rozov et al. BMC Bioinformatics 2014, 15(Suppl 9):S7
http://www.biomedcentral.com/1471-2105/15/S9/S7

BMC Bioinformatics

PROCEEDINGS Open Access

Fast lossless compression via cascading Bloom filters

Roye Rozov[1], Ron Shamir[1*], Eran Halperin[1,2,3]

From RECOMB-Seq: Fourth Annual RECOMB Satellite Workshop on Massively Parallel Sequencing
Pittsburgh, PA, USA. 31 March - 1 April 2014

Abstract

Background: Data from large Next Generation Sequencing (NGS) experiments present challenges both in terms of costs associated with storage and in time required for file transfer. It is sometimes possible to store only a summary relevant to particular applications, but generally it is desirable to keep all information needed to revisit experimental results in the future. Thus, the need for efficient lossless compression methods for NGS reads arises. It has been shown that NGS-specific compression schemes can improve results over generic compression methods, such as the Lempel-Ziv algorithm, Burrows-Wheeler transform, or Arithmetic Coding. When a reference genome is available, effective compression can be achieved by first aligning the reads to the reference genome, and then encoding each read using the alignment position combined with the differences in the read relative to the reference. These reference-based methods have been shown to compress better than reference-free schemes, but the alignment step they require demands several hours of CPU time on a typical dataset, whereas reference-free methods can usually compress in minutes.

Results: We present a new approach that achieves highly efficient compression by using a reference genome, but completely circumvents the need for alignment, affording a great reduction in the time needed to compress. In contrast to reference-based methods that first align reads to the genome, we hash all reads into Bloom filters to encode, and decode by querying the same Bloom filters using read-length subsequences of the reference genome. Further compression is achieved by using a cascade of such filters.

Conclusions: Our method, called BARCODE, runs an order of magnitude faster than reference-based methods, while compressing an order of magnitude better than reference-free methods, over a broad range of sequencing coverage. In high coverage (50-100 fold), compared to the best tested compressors, BARCODE saves 80-90% of the running time while only increasing space slightly.

Background

Deep sequencing has become almost ubiquitous in biology over the last decade. In the past five years, sequencing costs were halved every 5 months, while storage costs were halved every 14 months [1]. The long term effect of this trend is a growing gap between our capacity to store and analyze sequencing data, and our capacity to generate such data. For sharing results of large-scale experiments, the effects have already become readily apparent: physical hard disk transfer has become a common practice, and cloud analysis platforms have been embraced in order to avoid the prohibitive time requirements needed to download or store huge volumes.

As a result, much effort has been placed on representing sequencing data more compactly. Specialized compression tools tailored to this context have emerged, improving upon general purpose compressors, such as gzip. These tools fall into two categories - reference-based [2,3], and reference-free [4-6]. The former methods utilize knowledge of the genome from which reads were extracted (with mutations and errors), while the latter use no prior information. A recent article described the Pistoa Sequence Squeeze competition, wherein the relative merits

* Correspondence: rshamir@tau.ac.il
[1]Blavatnik School of Computer Science, Tel-Aviv University, Tel Aviv, Israel
Full list of author information is available at the end of the article

of many of these methods were compared. This article also introduced new high performance methods that were among the competition leaders [1].

Compression algorithms are evaluated by two main criteria: their compression ratio, namely, the ratios of compressed file sizes to original file sizes, and by their speed. In the context of compressing reads, compression ratios are often expressed in terms of the average number of bits per base for a fixed read length. Currently, reference-based methods generally compress most effectively, but require long run times. In order to compress reads, reference-based methods first call on a short-read aligner to find a best alignment position for each read. Such an alignment typically has only a few (or no) mismatches relative to the reference. Reads can then be represented as integers marking reference positions instead of as sequences, along with the set of differences relative to the reference. Further refinements can then be applied, such as sorting the reads by reference position and then encoding differences between consecutive positions to use fewer bits, and employing Huffman coding to encode more common mutations with less bits than rare ones [3,2]. Reference-free methods employ a variety of techniques, including boosting schemes for general purpose compressors [4,6], rough assembly for the sake of emulating reference-based compression [5], and arithmetic coding/context modeling approaches, which trade increases in run time for better compression ratios [1].

There is therefore an inherent tradeoff between runtime and compression ratio. Specifically, even though compression ratios are impressive for reference-based methods, their running times are often prohibitively high. In this work we propose a new method, Bloom filter Alignment-free Reference-based COmpression and DEcompression (BARCODE, abbreviated to BRC below), which achieves high compression ratios with a dramatic decrease of runtime. BARCODE does so by leveraging the space efficiency of Bloom filters, probabilistic data structures allowing queries of set membership. Their use has recently grown in popularity in bioinformatics [7,8,5,9], mainly to avoid the memory overhead needed to store large collections of k-length substrings of sequenced reads (k-mers) used to represent nodes of de Bruijn graphs in de novo assembly. To the best of our knowledge, this is the first use of Bloom filters for NGS compression.

Here, we adopt a similar scheme to that used for assembly in two recent works [8,10]. We hash whole reads into BFs as a means of compression. In tests performed, BARCODE's run times are closest to those reference-free methods while its compression ratios near those of reference-based methods. In as little as a ninth of the running time, we are able to compress to within less than 20% of the compression ratios observed for reference-based methods. We demonstrate that with higher coverage levels, BARCODE's efficiency improves, whereas reference-based methods show no improvement, while the gap in run time grows more severe. By comparing our method with several existing tools, we show its superior balance of speed and compression efficiency.

Methods

Technical background

A Bloom filter (BF) is an array A of size m having all positions initially marked 0. Elements are inserted into A by applying a collection of h hash functions: the output of each specifies a position to be marked with a 1 in A. Querying whether or not an element has been inserted involves applying the same h hash functions and checking the values at the positions they return. If at least one hash function returns 0, the element definitely was not inserted; if all 1s return, either it has been inserted, or it is a false positive. For a BF of size m, n entries can be inserted by h hash functions to achieve a false positive rate $F \approx (1 - e^{(-hn/m)})^h$. In [11], it is shown that for fixed m and n, F is minimized with $h = ln(2)\rho$, where $\rho = m/n$. Plugging this value back in for F leads to $F = c^\rho$, where $c = 0.6185$.

Encoding and decoding using a Bloom filter

Our method involves two basic processes: BF loading and querying. We initially assume all reads are unique and later relax this assumption. We load all reads into a BF B, and then use the reference genome to query it. We query B with read length subsequences (and their reverse complements) from all possible start positions on the genome. This allows us to identify all of the potential reads that correspond to genome positions, a set that covers most of the hashed reads. Some of the accepted reads will be false positives. In order to avoid them in the decoding process, we identify a set FP corresponding to all reads accepted by B that are not in the original read set. Additionally, since the reads are taken from a specimen whose genome contains mutations compared to the reference (and since sequencing is error-prone), some reads will not be recovered by querying the genome. We call this set of reads FN. FN and FP are stored separately from B, and compressed using an off-the-shelf compression tool. For a set of unique reads, this suffices to allow a complete reconstruction of the reads.

Decoding proceeds by decompressing B, FN, and FP, and then repeating the querying procedure. We initialize the read set to FN. Then, we query B with each position from the genome as done to identify elements of FP. Whenever B accepts we check if the accepted read is not also in FP, and add it to the read set if it isn't. To remove the unique read restriction, we first move all

repeated reads to *FN* before loading *B*. We treat reads containing 'N' characters similarly. These two additions allow us to circumvent an inherent limitation of Bloom filters - the loss of multiplicity information - and reduces the entropy in the (now multi-) set *FN*, making it more compressible. The encoding process with one BF is detailed in steps 1-4 of Figure 1 and Algorithm 1. The relative contributions of error reads and repeated reads to *FN* are discussed in the appendix.

Algorithm 1 Encode one **Input**: R, G; **Output**: B, FN, FP **Conventions**: Let g be the length of the reference genome G, q_i be the i^{th} genome query, ℓ_{read} be the sequenced read length, and P be the set of genome queries accepted by B. For brevity, we exhibit queries from only the forward strand, whereas our implementation queries (and accepts from) both strands.

FN := {$r : r \in R$ and (r is repeated in R or r contains an 'N')}
R' := R \ FN
for all $r \in R'$ do

insert(r, B)

end for
for all $i \in [1, g - \ell_{read} + 1]$ **do**
;if $qi \in B$ **then**
P := P ∪ q_i
end if

end for
FN := FN ∪ {R' \ P}
FP := P \ R'
return (*B, FN, FP*)

Encoding and decoding using a BF cascade

Although appealingly simple, we found the above method did not offer competitive compression, as the costs imposed encoding *FP* and *FN* outweighed the benefit of storing the unique reads in *B*. Thus, to reduce the number of false positives that need to be compressed separately, we use a cascade of BFs as in [10]. To this end, we rename *B* and *FP* above as B_1 and FP_1, respectively. We consider B_1 to be the first BF in a cascade, and each element of FP_1 is then hashed into a subsequent BF B_2. We note that since B_2 is meant to store false positive reads it should reject true reads: thus, any element of *R'* (the set of unique reads) accepted by B_2 is a false positive relative to B_2. Thus, to identify FP_2, we add each element accepted by querying *R'* against B_2. This process can be continued for any

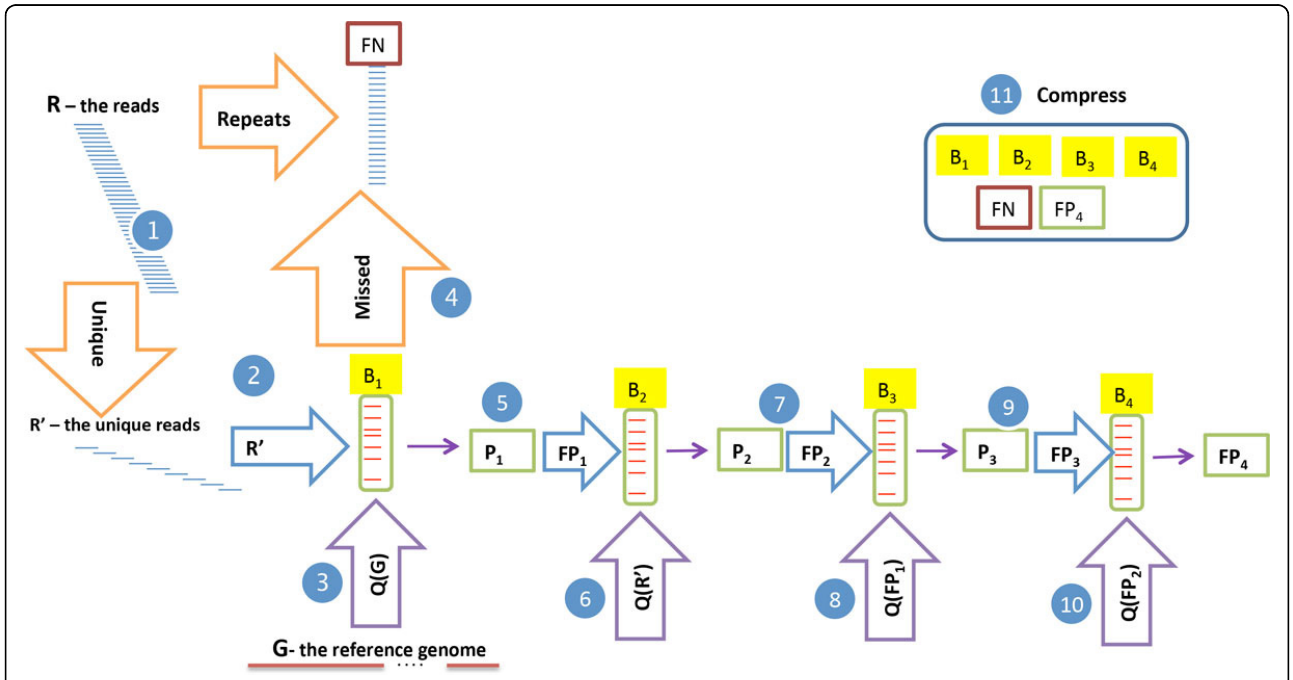

Figure 1 The encoding process. Step 1 separates the unique reads set *R'* from the repeated reads set *FN*. In step 2 unique reads (*R'*) are hashed into a BF *B*1 and the rest assigned to a set *FN*. In steps 3-4 all read-length sequences of the reference genome *G* are queried and reads accepted by B_1 that are not in *R'* are added to FP_1. Steps 5-10 show subsequent encoding via a BF cascade. False positives relative to each BF are input to the next BF. Each BF is then queried by using the set loaded into the last BF in the cascade. In step 11 additional compression is perfomed on the resulting BFs and sets. Orange arrows indicate assignments. Purple arrows marked with *Q*(.) indicate BF queries with sets denoted in parenthesis. Blue arrows indicate BF loading.

desired number of BFs. Once BFs are loaded in this way, to identify real reads, we query each BF in the cascade and accept reads only if the index of the first BF to reject them is even.

Since elements inserted to BF_j are necessarily a subset of those inserted to BF_{j-2}, we see an exponential drop-off in BF size (since F is fixed). Since sizes for successive BFs alternately depend on n and $(2g - n)\rho F \approx 2g\rho F$ (the number of false positives expected for $2g$ queries from G multiplied by the cost per element, assuming $g >> n$), we expect the total file size to be approximately $(n\rho + 2g\rho F + n\rho F + 2g\rho F^2 + ...)$ bits. Using $F = c^\rho$ from above, we observe that for an infinite cascade, the average number of bits per read is then

$$\left(\rho\frac{2g\rho c^\rho}{n}\right)(1 + c^\rho + c^{2\rho} + ...) = \left(1 + \frac{2gc^\rho}{n}\right)\left(\frac{\rho}{1 - c^\rho}\right). (1)$$

Here the left hand side represents the sum of costs due to the expected number of elements in each BF for an infinite cascade. In practice, we use four BFs and a numerical solver in scipy [12] employing the L-BFGS-B [13] algorithm to find the value of ρ minimizing the above expression. The small list FP_4 is encoded separately along with *FN*. The process is described in Figure 1 steps 5-11 and Algorithm 2. Decoding proceeds using queries from G as before, but in this case each accepted read is used to query subsequent BFs until rejection. This is depicted in Figure 2and Algorithm 3.

Algorithm 2 Encoding Let B_j be the j^{th} BF loaded ($j \in [2, 4]$) with FP_{j-1}, $S \cap B_j$ is short-hand notation for the subset of S accepted by B_j.

(B_1, *FN*, FP_1) := **Encode one**(R, G) # we initialize by calling Algorithm 1
F P_2 := R' ∩ B2
for j = 3 **to** j = 4 **do**
  for all $r \in FP_{j-1}$ **do**
    insert(r, B_j)
  end for
  $FP_j := FP_{j-2} \cap B_j$
end for
return (B_1, B_2, B_3, B_4, *FN*, FP_4)

Algorithm 3 Decoding **Input**: (B_1, B_2, B_3, B_4, *FN*, FP_4, G); **Output**: (R_{rc}) For brevity, reconstruction of only one strand is shown.
R_{rc} := FN
for all $i \in [1, g - \ell_{read} + 1]$ **do**
 for j = 1 **to** j = 4 **do**
  if $q_i \notin B_j$ **then**
   if j is even **then**
    $R_{rc} := R_{rc} \cup q_i$
   end if
  continue {increment i}
 end if
 end for
 if j = 4 **and** $q_i \in FP_4$ **then**
  $R_{rc} := R_{rc} \cup q_i$
 end if
end for
return R_{rc}

Additional compression

BF parameters are automatically set to make each BF more compressible. This involves incrementing the number of hash functions for each BF from 1 to the minimal number that allows it to both have an uncompressed file size lower than a preset threshold (we used 500 MB) and obtain the value of F from equation 1. Typically, this results in h being in the range of 1 to 3. We do this in order to reduce each BF's compressed size (at the expense of increasing its RAM occupation); this practice is introduced in [11].

Once BFs are loaded and the sets *FP*4 and *FN* are identified, we use 7zip [14] to compress the *B*1, ..., *B*4 and SCALCE [4] to compress the output lists FP_4 and *FN*. In principle, any general compression tool can be used for the BFs, and it is preferable to use a tool that takes advantage of existing sequence overlaps among the leftover reads to compress them efficiently.

Results and discussion

Comparison on simulated reads

We simulated reads from Human (hg19) chromosome 20 using dwgsim [15]. This tool introduces mutations into the reference genome and then samples reads from both genome strands using a user-defined per base error rate. We sampled 100 bp single end reads at various coverage levels with a 0.001 mutation rate and a per base error rate increasing from 0 to 0.005 from the 5' to the 3' end of reads (in line with current estimates of Illumina error rates [16]). We also demonstrated the effect of varying the error rate In Figure 4. All reported results were run on a 16 core AMD Opteron 6140 (2.6 GHz) 128 GB RAM server, running the Ubuntu 12.04 Linux operating system.

We found that BARCODE compresses more effectively at higher coverage. Although the proportion of reads in *FN* increases as the proportion of unique reads decreases (Table 1), BARCODE benefits from SCALCE's increasing efficiency due to greater redundancy among FNs. BARCODE's decode times were similar to its encode times, as would be expected since both rely on the same genome querying procedure.

To demonstrate that our use of BFs improves upon SCALCE's compression results, we compared our results with SCALCE run on all reads. We also tested quip [5] and fastqz [1], state-of-the-art tools in terms of both compression efficiency and speed [1]. All three tools

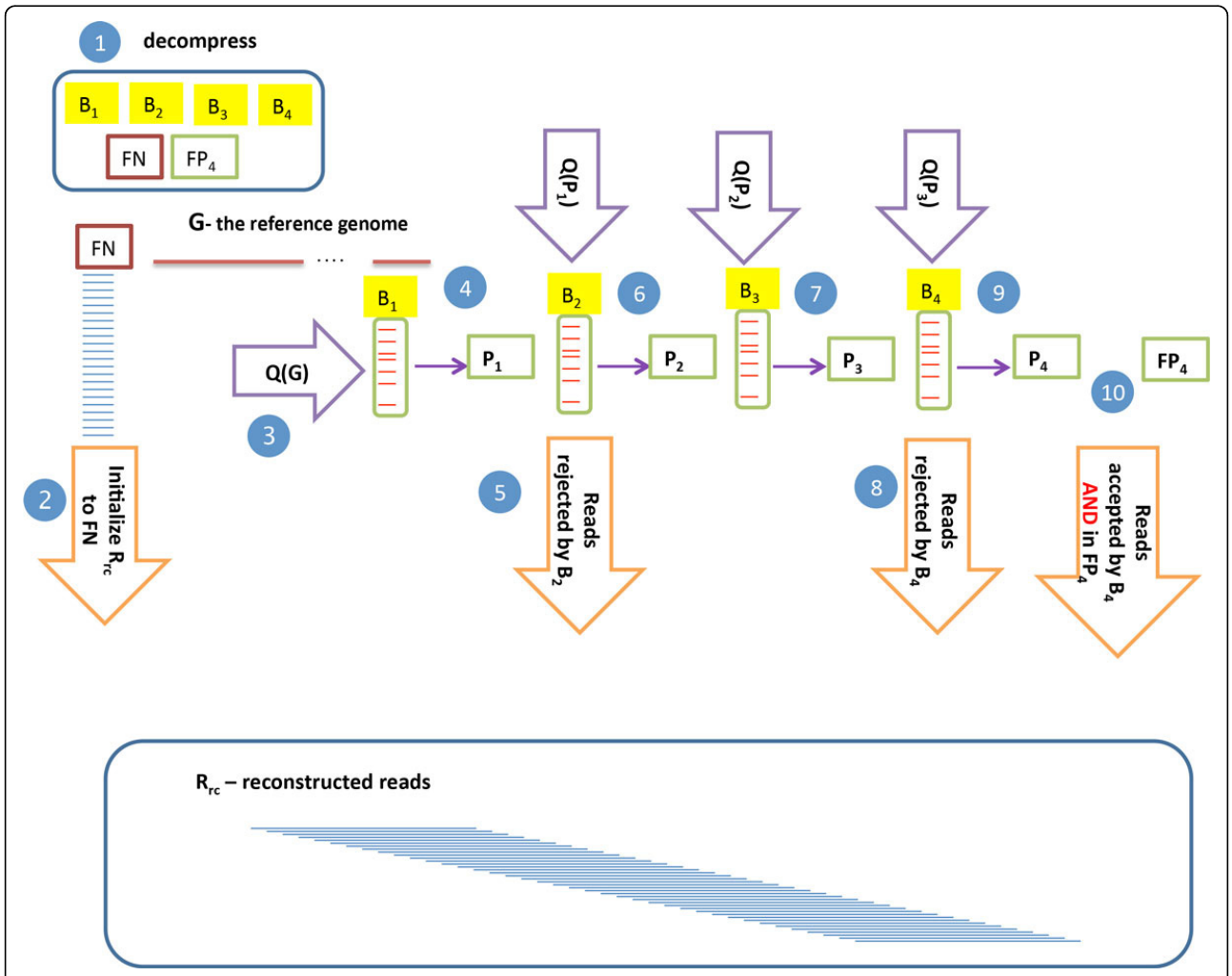

Figure 2 Decoding the reads. Following decompression of BFs, FP_4, and *FN*, BF querying commences. Each read accepted by a BF is used to query subsequent BFs until rejection. Reads rejected by even BFs or accepted by B_4 and in FP_4 are added to the reconstructed reads, R_{rc}. Purple arrows are consistent with Figure 1. Orange arrows indicate additions to R_{rc}, the reconstructed reads.

either output compression results or ratios separately for sequences, read names, and quality scores. We note that the best performers in the Sequence Squeeze competition in terms of base compression ratio, Sam-comp and CRAM, did not provide such outputs and thus did not allow direct comparison. Quip and fastqz also include both reference-based and reference-free modes. We performed alignment via bowtie2 [17] for quip runs while fastqz performed its own alignment. To ensure a fair comparison, all tools were run as a single thread when possible, including calls to 7zip and SCALCE from BARCODE. Fastqz used three threads during its run, as this

Table 1 BRC performance with varying coverage.

coverage	time (sec)	|*R*| (M)	|*FP4*| (K)	|*FN* | (M)	BF size (MB)	FP_4 size (MB)	*FN* size (MB)	compression bits/base
10	410	6.3	3.7	2.0	8.5	0.38	36.8	0.58
20	590	12.6	6.8	4.4	14.2	0.68	66.8	0.52
30	800	18.9	9.3	7.1	19.0	0.94	93.8	0.48
40	1006	25.2	20	10.2	22.8	2.01	119.0	0.46
50	1220	31.5	16	13.5	26.4	1.66	143.0	0.43

Reads were simulated from hg19 chromosome 20 with 100 bp single end reads. A mutation rate of 0.001 was used along with 0-0.005 per base error rate along the length of each read. Run times include additional compression steps performed by SCALCE and 7zip in single thread mode. *R* - the read set. FP_4 - the final false positive set. *FN* - the set of reads not encoded by the BFs.

was not a user-selectable parameter, but each thread was assigned to one of sequences, qualities, and names. Figure 3 compares BARCODE with other tools in terms of run time and compression efficiency. A full listing of program parameters used is provided in Table 2.

Overall, we found the compression ratio improved with greater coverage for all reference-free methods, and remained essentially constant for reference-based methods. Figure 3 shows that reference-based compressors are better in compression ratios but reference-free compressors are faster (An exception to this trend was fastqz, whose reference-based version is faster than its reference-free version, likely due to the use of context model-based arithmetic coding). Quip performed poorly in compressing sequences without a reference, showing it has apparently been optimized for speed and perhaps compression of qualities and read names. SCALCE shows strong dependence of compression ratio on coverage, as would be expected by its leverage of the recurrence of long subsequences. BARCODE takes advantage of this trend to also improve with higher coverage, even as the proportion of reads hashed to BFs decreases (See Table 1). BARCODE's times are closest to SCALCE and reference-free quip, and its compression ratios approach those of reference based methods, especially at higher coverage values. For most coverage values, it maintains an order of magnitude time advantage vs. reference-based methods (~2-3x vs. fastqz, ~5-7x vs. quip), as well as an order of magnitude compression advantage of the tested reference-free methods.

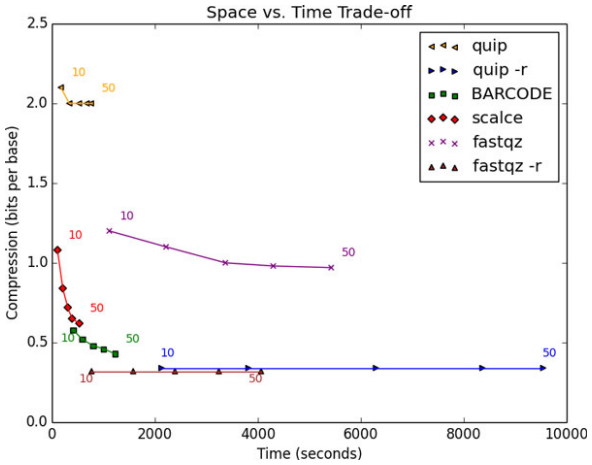

Figure 3 A comparison of sequence compressors. The figure shows elapsed real run time vs. compression ratios of read sequences in bits per base for read length 100 bp. The measurements of each method for different coverage levels are connected by a line. Points correspond to coverage levels from 10 to 50 in multiples of 10 from left to right. Methods denoted with an "-r" were run with the reference-based option. Run times were measured with /usr/bin/time using a single thread on the same Linux server.

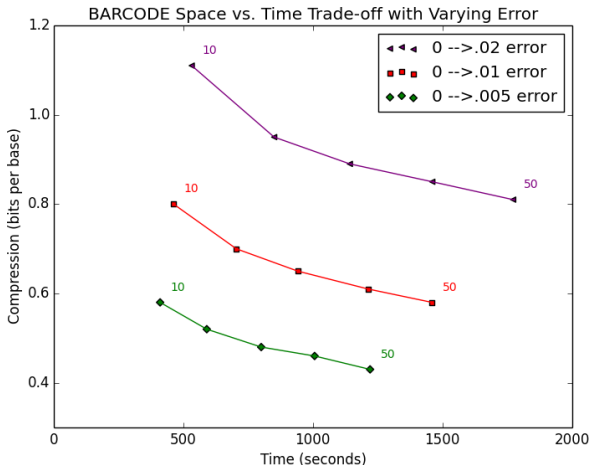

Figure 4 The effect of varying error rate. BARCODE runs are shown with error rates varying from 5' to 3' ends as indicated in the figure legend. Compression ratios increase with greater error, but higher coverage compensates somewhat. Overall, as coverage increases, run time is effected by increasing error more than compression ratios, as can be seen by the decreasing slopes between fixed coverage points between the green and red curve and between the red and purple curves.

Higher coverage, longer reads

We tested scenarios of higher coverage and longer read lengths: (1) coverage 100 and read length 100 bp, (2) coverage 100 and read length 200 bp, and (3) coverage 200 and read length 400 bp. Table 3 shows a continuation of the trends expressed at lower coverage levels. Higher coverage aided reference-free methods, but not reference-based methods. Longer reads improved compression ratios in each case with the exception of fastqz -r. We observed larger impacts on run time as a result of doubling read length than coverage.

Conclusions

We have presented a new approach to compressing sequencing reads, bridging the gap between the speed of

Table 2 Program parameters used in compression tool comparison (Figure 3)

Program	Parameters
dwgsim	-C coverage level -H -e 0.0-0.005 -R 0.0 -1 read length -2 0 -y 0.0
bowtie2	-x chr20 -U input fastq -S
SCALCE	input fastq -T 1 -A -n library -o output prefix
quip (default)	-o=quip -i=sam input sam
quip (reference)	-o=quip -r ref fa -i=sam input sam
fastqz (default)	c input fastq output prefix
fastqz (reference)	c input fastq output prefix r packed ref file
BRC	rec load bf -err rate 0 -e 0 -i 4 reads file packed ref file

Table 3 Performance comparison on high coverage values and read lengths longer than 100.

Time (sec)	coverage	quip	scalce	fastqz	BRC	quip -r	fastqz -r
	50	759	533	5426	1220	9544	4063
	100	1599	1016	10417	2211	20216	7479
	100, len 200	1280	1165	8760	1400	21705	7400
	200, len 400	2284	2518	15706	2209	51628	17769
Compression (bits/base)	50	2.0	0.62	0.97	0.43	0.34	0.32
	100	2.0	0.53	0.58	0.40	0.34	0.32
	100, len 200	1.8	0.43	0.61	0.37	0.19	0.45
	200, len 400	1.7	0.32	0.41	0.31	0.12	0.41

Reads were generated as described in the main text. Quip -r times include bt2 alignment.

alignment-free, reference-free methods and the compression efficiency of reference-based methods. We have tested the dependence of extant sequence compressors on coverage levels and shown that while reference-based methods compress most efficiently, they place a heavy burden on CPU times due to alignment and cannot leverage added redundancy to benefit compression ratios. Reference-free methods do benefit from higher coverage, but maintain a considerable distance from reference-based methods in terms of compression ratios even at the highest levels tested. Although we have shown that our new method, BARCODE, obtains a better trade-off than either of these extremes, we maintain that there remains much room for improvement, even when considering the inherent constraints imposed by the Kolmogorov complexity of the data. We note that further comparison to other methods like CRAM [3] and sam_comp [1] is needed.

BARCODE is currently a proof-of-principle implementation, and thus we expect that further optimization will improve run time and compression efficiency. Compression ratios may be improved by taking advantage of better general compression tools available such as the ZPAQ library [18], as fastqz and sam_comp do. Thus far, we have not utilized arithmetic coding techniques because they employ multiple threads and thus introduce significant additional resource requirements. Our approach can also be extended to allow for fast access to variants in the original data by using conventional BFs that are not compressed, and by compressing FN/FP reads using encoding that allows fast random access (at some expense of compression ratio). We aim to investigate these paths in the future.

Appendix

Real data test

We examined BARCODE's performance on the C. Elegans data set tested in the Sequence Squeeze competition, SRR065390_1. This data set consists of 33415360 100 bp reads, amounting to 33-fold coverage of the genome. BARCODE's compression ratio on this data was 0.46 bits per base, and run time was 1203 seconds, in line with experiments described in the main text and comparable with reference-based methods tested in the Sequence Squeeze competition [1].

Contributions of repeated reads vs. errors to FN

FN is comprised of repeated reads filtered out to preserve their multiplicities, and reads differing from the reference because of errors or variations. Here, we describe the relative contributions of each part. The expected number of repeated reads can be described probabilistically. Assuming reads are sampled independently from G, given a read r, the probability of drawing r again is $1 - \frac{1}{G}$. For R reads, the probability of observing no repetitions is then $\left(1 - \frac{1}{G}\right)^{R-1}$. Thus, the expected number of repeated reads is $R\left(1 - \left(1 - \frac{1}{G}\right)^{R-1}\right)$. Since we hash reverse complement reads from reference strands separately, we revise the length considered to $2G$. Since we wish to count the total multiplicity of each repeated read, the contribution of repeated reads to FN is thus approximated by $2R\left(1 - \left(1 - \frac{1}{2G}\right)^{R-1}\right)$. Clearly, this contribution depends on coverage, as shown in Table 4.

We model the contribution of error to FN using $Binom(100, p)$ with $p = 0.0025$, the mean error over the read length used in our simulated reads (where error varies from 0 to 0.005 from the 5' to 3' ends). Most of

Table 4 Counts of repeats vs. errors with increasing coverage. The proportion of reads due to errorsremains roughly constant, while the proportion due to repeats increases as coverage increases.

Coverage	$\|R\|$ (M)	repeats (M)	$\|FN\|$ (M)
10	6.3	0.3	2.0
20	12.6	1.3	4.4
30	18.9	2.8	7.1
40	25.2	4.9	10.2
50	31.5	7.4	13.5

errors with increasing coverage. The proportion of reads due to errors remains roughly constant, while the proportion due to repeats increases as coverage increases.

the mass is carried by the one and two error terms, leading to a relative error proportion estimate of $\binom{100}{2}(1-p)^{98}p^2 + \binom{100}{2}(1-p)^{99}p$. Table 4 shows this proportion is independent of coverage level.

Availability

BARCODE can be downloaded at http://www.cs.tau.ac.il/~heran/cozygene/software.shtml.

List of abbreviations

BF- Bloom filter; BRC - BARCODE; bpb - bits per base; bp - base pairs

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

RR developed the method. RS and EH designed the experiments. RR implemented the method and performed experiments. All authors analyzed results, co-wrote the manuscript, and read and approved the final manuscript.

Acknowledgements

RR would like to thank Oron Navon and Roy Ronen for helpful comments in preparation of the manuscript.

Declarations

RS was supported in part by the Israel Science Foundation (grant 317/13) and by the Raymond and Beverly Sackler chair in bioinformatics. RR was supported in part by a fellowship from the Edmond J. Safra Center for Bioinformatics at Tel-Aviv university, and by the Center for Absorption in Science, the Ministry of Immigrant Absorption in Israel. EH is a faculty fellow of the Edmond J. Safra Center for Bioinformatics at Tel Aviv University. EH was partially supported by the Israeli Science Foundation (grant 1425/13), and by National Science Foundation grant III-1217615.
This article has been published as part of BMC Bioinformatics Volume 15 Supplement 9, 2014: Proceedings of the Fourth Annual RECOMB Satellite Workshop on Massively Parallel Sequencing (RECOMB-Seq 2014). The full contents of the supplement are available online at http://www.biomedcentral.com/bmcbioinformatics/supplements/15/S9.

Authors' details

[1]Blavatnik School of Computer Science, Tel-Aviv University, Tel Aviv, Israel. [2]Molecular Microbiology and Biotechnology Department, Tel-Aviv University, Tel Aviv, Israel. [3]International Computer Science Institute, Berkeley, CA, USA.

Published: 10 September 2014

References

1. Bonfield JK, Mahoney MV: **Compression of FASTQ and SAM format sequencing data.** *PloS One* 2013, **8(3)**:59190.
2. Kozanitis C, Saunders C, Kruglyak S, Bafna V, Varghese G: **Compressing genomic sequence fragments using SlimGene.** *Journal of Computational Biology* 2011, **18**:401-413.
3. Hsi-Yang Fritz M, Leinonen R, Cochrane G, Birney E: **Efficient storage of high throughput DNA sequencing data using reference-based compression.** *Genome Research* 2011, **21**:734-740.
4. Hach F, Numanagic I, Alkan C, Sahinalp SC: **SCALCE: boosting sequence compression algorithms using locally consistent encoding.** *Bioinformatics* 2012, **28(23)**:3051-7.
5. Jones DC, Ruzzo WL, Peng X, Katze MG: **Compression of next-generation sequencing reads aided by highly efficient de novo assembly.** *Nucleic Acids Research* 2012, **40(22)**:171.
6. Cox AJ, Bauer MJ, Jakobi T, Rosone G: **Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform.** *Bioinformatics* 2012, **28(11)**:1-6.
7. Pell J, Hintze A, Canino-Koning R, Howe A, Tiedje JM, Brown CT: **Scaling metagenome sequence assembly with probabilistic de Bruijn graphs.** *Proceedings of the National Academy of Sciences* 2012, **I(1)**:1-11.
8. Chikhi R, Rizk G: **Space-efficient and exact de Bruijn graph representation based on a Bloom filter.** *Algorithms in Bioinformatics* 2012, 236-248.
9. Melsted P, Pritchard J: **Efficient counting of k -mers in DNA sequences using a bloom filter.** *BMC Bioinformatics* 2011, **12**:333.
10. Salikhov K, Sacomoto G, Kucherov G.: **Using cascading bloom filters to improve the memory usage for de Brujin graphs.** In *Algorithms in Bioinformatics Lecture Notes in Computer Science* Darling, A., Stoye, J 2013, **8126**:364-376.
11. Mitzenmacher M: **Compressed Bloom filters.** *IEEE/ACM Transactions on Networking* 2002, **10.**
12. Oliphant TE: **SciPy: Open source scientific tools for Python.** *Computing in Science and Engineering* 2007, **9**:10-20.
13. Zhu C, Nocedal J, Byrd RH, Lu P: **Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization.** 1997.
14. Pavlov I: **7zip compression software.** [http://www.7-zip.org].
15. Homer N: **Dwgsim read simulations software.** [https://github.com/nh13/dwgsim].
16. Loman NJ, Misra RV, Dallman TJ, Constantinidou C, Gharbia SE, Wain J, Pallen MJ: **Performance Comparison of Benchtop High-Throughout Sequencing Platforms.** *Nature Biotechnology* 2012, **30**:434-9.
17. Langmead B, Salzberg SL: **Fast gapped-read alignment with Bowtie 2.** *Nature Methods* 2012, **9**:357-359.
18. Mahoney M: *ZPAQ compression software* [http://mattmahoney.net/dc/zpaq.html].

doi:10.1186/1471-2105-15-S9-S7
Cite this article as: Rozov *et al.*: Fast lossless compression via cascading Bloom filters. *BMC Bioinformatics* 2014 **15**(Suppl 9):S7.

Chapter 3

Recycler: an algorithm for detecting plasmids from de novo assembly graphs

Bioinformatics, 33(4), 2017, 475–482
doi: 10.1093/bioinformatics/btw651
Advance Access Publication Date: 24 November 2016
Original Paper

OXFORD

Genome analysis

Recycler: an algorithm for detecting plasmids from *de novo* assembly graphs

Roye Rozov[1], Aya Brown Kav[2], David Bogumil[2], Naama Shterzer[2], Eran Halperin[1,3,4], Itzhak Mizrahi[2,*] and Ron Shamir[1]

[1]Blavatnik School of Computer Science, Tel-Aviv University, Tel Aviv, Israel, [2]The Department of Life Sciences & the National Institute for Biotechnology in the Negev, Ben-Gurion University of the Negev, Beer-Sheva, Israel, [3]Molecular Microbiology and Biotechnology Department, Tel-Aviv University, Tel Aviv, Israel and [4]International Computer Science Institute, Berkeley, CA, USA

*To whom correspondence should be addressed.

Associate Editor: Prof. Alfonso Valencia

Received on June 25, 2016; revised on September 22, 2016; accepted on October 9, 2016

Abstract

Motivation: Plasmids and other mobile elements are central contributors to microbial evolution and genome innovation. Recently, they have been found to have important roles in antibiotic resistance and in affecting production of metabolites used in industrial and agricultural applications. However, their characterization through deep sequencing remains challenging, in spite of rapid drops in cost and throughput increases for sequencing. Here, we attempt to ameliorate this situation by introducing a new circular element assembly algorithm, leveraging assembly graphs provided by a conventional *de novo* assembler and alignments of paired-end reads to assemble cyclic sequences likely to be plasmids, phages and other circular elements.

Results: We introduce Recycler, the first tool that can extract complete circular contigs from sequence data of isolate microbial genomes, plasmidome and metagenome sequence data. We show that Recycler greatly increases the number of true plasmids recovered relative to other approaches while remaining highly accurate. We demonstrate this trend via simulations of plasmidomes, comparisons of predictions with reference data for isolate samples, and assessments of annotation accuracy on metagenome data. In addition, we provide validation by DNA amplification of 77 plasmids predicted by Recycler from the different sequenced samples in which Recycler showed mean accuracy of 89% across all data types—isolate, microbiome and plasmidome.

Availability and Implementation: Recycler is available at http://github.com/Shamir-Lab/Recycler

Contact: imizrahi@bgu.ac.il

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Plasmids are extra-chromosomal DNA segments carried by bacterial hosts. They are usually shorter than host chromosomes, circular and encode nonessential genes. These genes are responsible for either plasmid-specific roles such as self-replication and transfer, or context-specific roles that can be beneficial or harmful to the host depending on its environment. Along with viruses and transposable elements, plasmids are members of the group termed mobile genetic elements (Doring and Starlinger, 1984) as they transmit genes and their selectable functions between microbial genomes. Plasmids play a central role in horizontal gene transfer (Halary *et al.*, 2009), and thus genome innovation and plasticity—fundamental forces in microbial evolution. Much interest has recently arisen for plasmid extraction and characterization, in particular because of their known roles in antibiotic resistance and in increasing metabolic outputs of agricultural or industrial byproducts. For instance, antibacterial resistance genes encoded on plasmids have long been known as a major issue for human health in clinical practice (Neu, 1992), but are also one of today's standard tools in microbiology and genetics

when used to select for specific cells (Bevan *et al.*, 1983). In order to derive plasmid sequences (which may be known or novel), one may choose from the following approaches: sequence already isolated microbes with their residing plasmids, sequence the overall microbial community of genomes (termed metagenome) from some environment or, as was recently described, sequence only the overall plasmid fraction from a given environment [termed plasmidome (Brown Kav *et al.*, 2012, 2013)]. The first technique obtains a mixture of chromosomal and plasmid DNA occurring together in a single strain. Since sequenced reads are devoted to only a few different sequenced DNA elements (the genome in question or any of its mobile elements), each is expected to be highly covered, and thus for species having low repeat content a good assembly can be achieved.

For natural environments containing many elements, often including those that are difficult to culture (Gilbert and Dupont, 2011) in a lab, metagenome assembly is attempted. This technique allows a much broader view of all taxa present and their plasmids, but is limited in that the characterization of each individual element depends on its coverage in the mixed DNA sample and the frequency of co-occurring repeats shared among different elements of the sample. Resulting assembled genomes of elements that are rare in the environment are thus often fragmented, and very high coverage (Howe *et al.*, 2014) is needed for accurately assembling them. However, assembly of metagenomes remains a highly active area of research: current assembly outputs are lacking and do not represent the true genetic capacity and synteny of genomes present in complex microbial communities. Since most of the DNA in these environments is due to host genomes, this approach currently provides only limited resolution of plasmids.

Most recently, a third technique has emerged that allows recovery of far greater numbers of plasmids. Plasmidome sequencing (Brown Kav *et al.*, 2012, 2013; Jørgensen *et al.*, 2014) allows nearly all sequencing resources to be devoted to circular DNA. Using a protocol described in (Brown Kav *et al.*, 2012), chromosomal DNA is filtered out and circular DNA segments are selectively amplified. Based on this protocol, hundreds of new plasmids were identified in the cow rumen (Brown Kav *et al.*, 2013) and rat cecum (Jørgensen *et al.*, 2014). Jørgensen *et al.*, (2014) applied the protocol introduced in Brown Kav *et al.* (2012) combined with bioinformatic validation of circularity. This post-assembly analysis resulted in a 95% PCR validation rate out of 40 randomly selected assembled contigs. This success raises the prospect of *in silico* refinement of plasmids beyond the initial assembly. Although Jørgensen *et al.*'s method was shown to have a high validation rate, its output is limited by the contiguity of the underlying assembler's contigs [in their case IDBA-UD (Peng *et al.*, 2012)], because it provides no means of combining multiple overlapping contigs to form cycles. It is a filtering process meant to identify probable circular sequences among sequences already output by the assembler. To date, no tools for plasmid assembly from short reads have been introduced to address these limitations.

In all of the above approaches plasmid assembly is hindered by several inherent characteristics derived from their mobile nature. These characteristics include their tendency to carry repetitive elements such as insertion sequences and to share genes with other plasmids and microbial genomes. In the context of *de novo* assembly, repeats cause collapse of linear sequences sharing them as subsequences. This creates ambiguity in the sense that it becomes unclear which extensions entering the repeat should be paired with those exiting it, where sequences begin and end, and whether there are unique terminal points at all as opposed to the sequence being circular. *De novo* assembly for the sake of identifying plasmids can be augmented by long-read sequencing (Conlan *et al.*, 2014; Hunt *et al.*, 2015) because such reads may be sufficiently long to bridge repeats short reads cannot. However, this approach is primarily limited to isolates or low complexity environments. This is evident in that long reads often depend on single molecule sequencing without amplification, thus only capturing relatively abundant DNA fragments. Besides repeats, chimeric sequences also present significant challenges to assembly, in that they create false connections between sequences and thus may lead to mis-assemblies.

To overcome some of these challenges, Antipov *et al.*, (2016), introduced plasmidSPAdes, an extension of the SPAdes assembler (Bankevich *et al.* (2012) that identifies likely 'plasmid components' in isolate whole genome sequencing experiments. This method looks for long contigs in the assembly graph that are sufficiently different coverage from those of the host genome. Here, we take a different approach to improve discovery of sequenced plasmids. We similarly analyze assembly graphs, but consider all nodes instead of paring the graph around long contigs. In addition to coverage, we also incorporate paired-end read mappings and topology, only reporting cycles when there is sufficient evidence that they are physically separate entities. We also accept as input any assembly graph, making our method applicable to isolate as well as metagenome and plasmidome samples.

Our inputs are an assembly graph $G=(V,E)$, and the mapping of paired-end reads responsible for the assembly to its nodes. The set of nodes V are sequences having associated lengths and coverage levels, and the set of arcs E is composed of directed connections among the nodes. Arcs are the result of branch points in the underlying de Bruijn graph: a branch node has outgoing arcs to two (or more) different nodes based on overlaps, and in many cases, the assembler does not have a definite way of choosing which extension is true in order to simplify the branch into a linear path. We aim to generate a set of putative cycles that are likely to be plasmids, and assign a coverage level for each one.

After defining this problem formally below, we present an algorithm (and its implementation) designed to address it, called Recycler. Recycler leverages assembly graphs output by SPAdes to specifically enable *de novo* assembly of plasmids and other cyclic sequences likely to be physically separated from the rest of the sequences present. We show it greatly improves recovery of plasmids over naive assembly and alternative methods, namely Jørgensen's and SPAdes' built-in repeat resolution, introduced in (Prjibelski *et al.*, 2014) and performs similarly to plasmidSPAdes on isolate sample inputs. We demonstrate Recycler's performance by applying it on both simulated and real data. We find that Recycler greatly increases recall while maintaining high precision. This is established via comparisons performed on simulated plasmidomes of various sizes. We also show that Recycler can be applied for plasmid assembly on real data from a bovine rumen plasmidome and metagenome, and from two different *Escherichia coli* isolate strains. In the isolate cases, Recycler recovered most known plasmids, and predicted additional sequences that matched known mobile elements from different hosts—all of which were identical or nearly identical to known reference sequences. In all cases on real data, Recycler either matched or exceeded the proportion of outputs matching plasmid annotation, as described in Brown Kav *et al.* (2013).

1.1 Related work

We note plasmid assembly is a multi-assembly problem, as described in the context of RNA-Seq transcriptome assembly (Pertea *et al.*, 2015). Formulations of such problems often aim to generate a minimal set of paths that maximize agreement with observed data

(Pertea *et al.*, 2015; Tomescu *et al.*, 2013; Trapnell *et al.*, 2010). These methods usually employ network flow formulations, which admit polynomial-time algorithms for minimizing flow cost on the network; this flow corresponds to a convex function of the sum of coverage differences between observed and estimated coverage levels. However, these methods resort to heuristics in selecting a minimal set of paths to cover the entire graph, as splitting a flow into a minimal number of path and cycle components is an NP-hard problem (Hartman *et al.*, 2012).

Recycler does not aim to generate a set of paths explaining all coverage levels, and thus does not depend on a global objective function encompassing all nodes or edges. This approach is avoided because of the presence of linear paths due to either plasmids not fully covered during sequencing or bacterial host genomes housing plasmids, which may introduce noise into coverage levels observed and will not be part of the solution. Avoiding a global objective imposing parsimony on paths also allows Recycler to benefit from a polynomial time algorithm for generating 'good' cycles. Thus, Recycler's approach is similar to StringTie (Pertea *et al.*, 2015), in that both repeatedly seek locally best paths or cycles and use coverage levels estimated on those to update coverage levels on the original graph, until some stopping criterion is met. We note the set of cycles desired is explicitly not minimal, as in cycle cover formulations (Gross *et al.*, 2013). For example, given a figure 8 component (Supplementary Figure S1, panel I), Recycler may represent it as two cycles separated by distinct coverage levels, where a minimal cover would use only one cycle. Instead, we wish to cover as much of the graph as possible with 'good' cycles.

2 Methods

2.1 Overview of recycler

The inputs to Recycler are a FASTG file representing a directed graph with vertices corresponding to non-branching sequence contigs and edges corresponding to connecting overlapping k-mers, and a BAM file of paired-end read mappings to the graph's nodes. The graph can be viewed as a compacted de Bruijn graph starting from order k of the sequence data by contracting edges (u, v) whenever u has outdegree 1 and v has indegree 1, and the sequence contig of the new node replacing u and v is the concatenation of their sequences. Each node has a coverage value reflecting its abundance in the input sequences. We search for cycles in the graph that will correspond to plasmids. Cycle sequence length, number of vertices and coverage uniformity are factored in the selection process. We also use paired-end read mappings including mates on different nodes as a proxy for which of the nodes may have emerged from the same physical DNA fragment. This provides a means of inferring whether a candidate cycle is a plasmid or a genomic segment including repeats that lead to ambiguous cycles in the graph. Once a best cycle is selected, its latent coverage level is determined and subtracted from those of all participating nodes. Nodes whose resulting coverage values become non-positive are then removed from the graph, allowing only those with some remaining coverage the opportunity to take part in additional cycles. Hence, the whole process can be viewed as greedily 'peeling off' cycles from the graph. Ideally, one would like the process to end in an empty graph, in which case the input graph would be exactly the union of the cycles found. In reality, the process is stopped when quality criteria for new cycles in the remaining graph are unmet.

2.2 Notations and definitions

Our input is a directed graph $G = (V, E)$, where V is a set of linear sequences having either a branch-point or terminal k-mer at each end and no internal branch-points. E is the set of overlaps between nodes, where $E = \{(u,v)$: the $(k-1)$-mer suffix of u = the $(k-1)$-mer prefix of $v\}$. We call a node *simple* if its indegree and outdegree are 1. A node v corresponding to sequence s of length $l(s)$ is assigned two positive values, $len(v)$ and $cov(v)$. $len(v) = l(s) - k + 1$ is called the length of the node (the subtraction is in order avoid double-counting bases common to overlapping segments at their ends). $cov(v)$, its coverage, reflects the average number of times each k-mer in s appears in the input read data. The input can be produced by a short read assembly tool. We further assign a weight $w(v) = \frac{1}{len(v)cov(v)}$ for each node v, resulting in low weight for high coverage and long nodes. Longer contigs tend to be less prone to random fluctuations in coverage, and are thus more reliable coverage indicators. For each cycle c in the graph, we assign each node a value representing its length fraction in c: $f(c, v) = \frac{len(v)}{\sum_{v' \in c} len(v')}$. The value $f(c, v)$ is used to define the mean and standard deviation of weighted coverage of cycle c as $\mu(c) = \sum_{v \in c} f(c, v)cov(v)$ and $STD(c) = \sqrt{\sum_{v \in c} f(c, v)(cov(v) - \mu(c))^2}$, respectively, and consequently the coefficient of variation of c, $CV(c) = \frac{STD(c)}{\mu(c)}$. $CV(c)$ is used to allow direct comparison of variation levels between cycles, independently of the magnitude of coverage of each. $CV(c)$ is indicative of coverage uniformity along c, and plasmids are expected to have uniform coverage levels that in many cases are different from other plasmids and their hosts. Thus, cycles with low CV values are more likely to correspond to plasmids than cycles with high CV values.

2.3 Our approach

Intuitively, plasmids should form cycles that are distinctive from the rest of the graph and have near uniform coverage. We also expect plasmid cycles to include few nodes, as each additional node introduced for a fixed sequence length increases fragmentation and the tendency of nodes to be common to more than one path. With this in mind, we search for 'good cycles' in the graph that potentially correspond to plasmids. Formally, we define a good cycle as a simple cycle in the graph satisfying the following constraints:

1. Minimum path weight for some edge: $\exists (u, v) \in c$ such that $c \backslash (u, v)$ (the path obtained by removing (u, v) from c) is a minimum weight path (by sum of weights $w(v)$) from v to u.
2. Low coverage variation: $CV(c) \leq \frac{\tau}{|c|}$, where τ is a defined threshold and $|c|$ is the number of nodes in.
3. Concordant read mapping: For pair r_1, r_2 of paired-end mates, if r_1 maps to a simple node in c then r_2 must also map to some node in c.
4. Sufficient sequence length: $\sum_{v \in c} len(v) \geq L$, where L is a defined threshold.

The first constraint is critical in order to avoid merging of two or more plasmids that are connected through a repeated region (Supplementary Figure S1, panel I). In addition, lower weight cycles correspond to longer sequence length and higher coverage nodes, and tend to have fewer nodes. Furthermore, for each edge this constraint uniquely determines at most one cycle that passes through the edge, thus avoiding consideration or enumeration of an exponential number of possible cycles. We note there are special cases allowing for cycles that visit a single node more than once; such a

case is shown in Supplementary Figure S1, panel II. The second constraint ensures that the coverage variation is low, thus again increasing our confidence that the cycle corresponds to exactly one plasmid. Moreover, this constraint implicitly ensures high coverage cycles, since low coverage cycles tend to have higher CV value. The third constraint exploits paired-end reads. Suppose we have a read pair r_1, r_2 and r_1 maps to a certain node in the candidate cycle c. We expect r_2 to map to the same cycle, unless r_1 falls on a node that is common to c and some other path p overlapping with it. In that case r_2 may map p to as well. Simple nodes are less likely to overlap with several cycles and paths, and the third constraint leverages this observation. We waive this constraint in case the coverage of c is sufficiently high, as in such cases the cycle 'stands out' from the background coverage. See Supplementary Material for details.

The above definition of a good cycle provides a mechanism for the identification of putative plasmids. Recycler processes each strongly connected component separately. It repeatedly finds a good cycle with minimum CV value, assigns it *latent coverage* equal to the mean cycle coverage and subtracts that coverage from the graph, creating a new *residual coverage* (Fig. 1). The weights of the vertices in the cycle are updated based on their new coverage values, and vertices whose resulting coverage values become non-positive are removed from the graph, allowing only those with positive residual coverage the opportunity to take part in additional cycles. After each such change, cycles are recalculated the same way using the updated coverage levels. This process continues as long as new good cycles are found. To avoid examining a potentially exponential number of cycles, we consider one minimum weight cycle through each edge in the graph. The algorithm selects the cycle with the lowest CV among these minimum weight cycles and 'peels it off' the graph. Algorithm 1 sketches the procedure for a single component. See the Supplementary Material for additional details.

2.4 Complexity

Algorithm 1 presented above terminates in polynomial time. In each iteration, if any good cycles exist, one is chosen and its mean coverage is calculated. There is at least one node in the cycle with coverage smaller than the mean coverage of the cycle, which is subsequently removed from the graph. Therefore, in each iteration at least one node is removed, and the number of iterations is bounded by the number of nodes. Using Johnson's algorithm (Johnson, 1977), the runtime of each iteration is $O(|V|^2 \log(|V|) + |V||E|)$. Running times are further reduced by

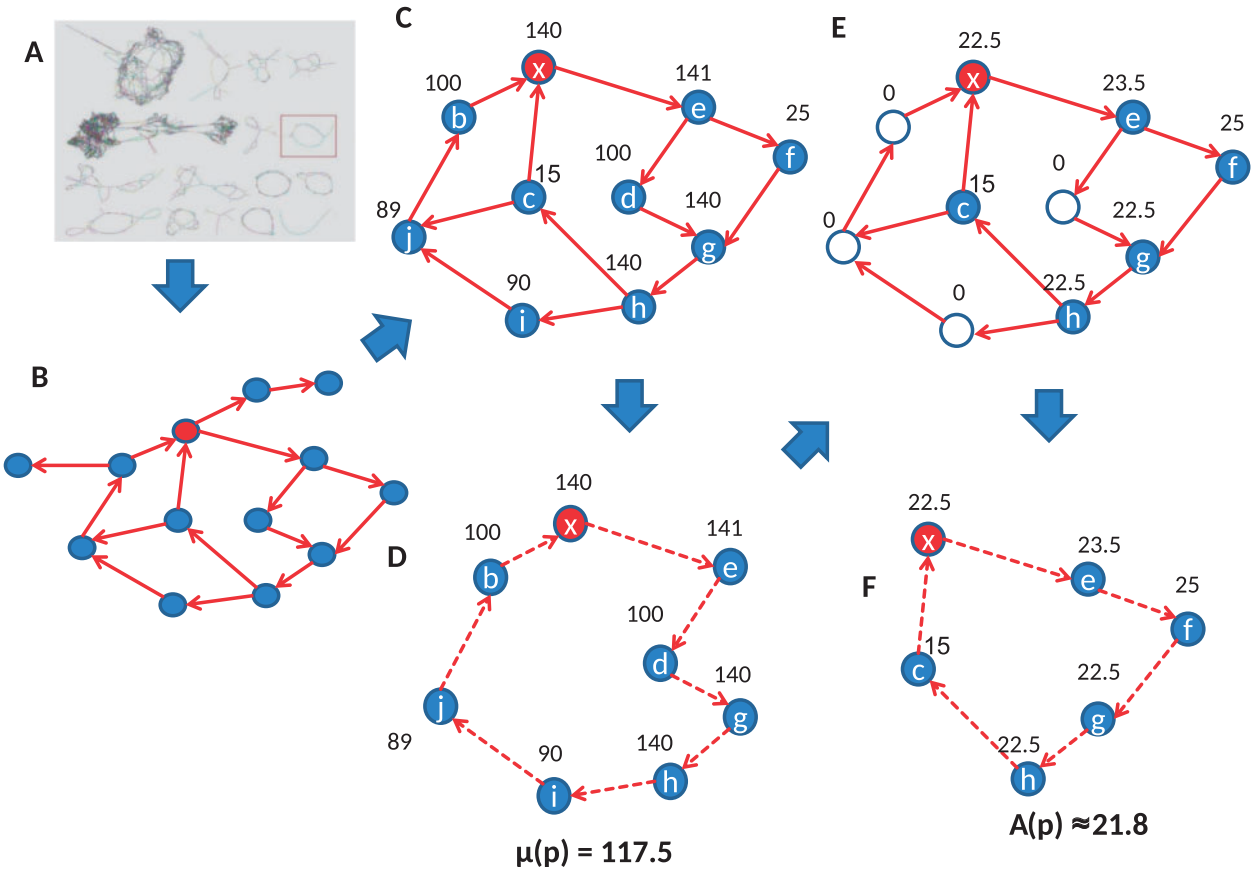

Fig. 1. Recycler work-flow. An example is shown of generating candidate cycles and peeling off cycles iteratively. For simplicity, all lengths are assumed to be equal and not shown. Here, we consider only candidate cycles that pass through vertex *x*, but ordinarily such candidates would be generated for each vertex in the component, and the cycle with lowest CV will be chosen and peeled off. (**A**) The assembly graph. (**B**) A single component is selected from the assembly graph (framed in A) and represented with vertices for contigs and edges for connecting *k*-mers. (**C**) The reduced component after tip removal. The numbers next to vertices are their observed contig coverage. Since vertex *x* has two incoming edges from vertices *b* and *c*, two candidate cycles are generated that pass through edges (*b*, *x*) and (*c*, *x*), respectively. This is done by computing shortest paths from *x* to *b* ($x, e, d, g, h, i, j, b, CV = 0.20$, shown in D) and from *x* to *c* ($x, e, d, g, h, c, CV = 0.41$, not shown). Two successive steps of peeling cycles are shown with their respective latent coverage assignments. First, the cycle in D is peeled off because the CV calculated from initially observed coverage is lowest for this cycle. Uncolored vertices correspond to contigs with zero coverage that are removed

Algorithm 1: Finding good cycles and peeling them off each component

Data: $G = (V, E, len, cov, w), \tau, L$
Result: Σ, the set of cycles
Compute shortest cycles passing through each edge;
for *each edge (u, v)* **do**
  Compute a minimum weight path p from v to u, if one exists;
  Compute the CV of the cycle $(p, (u, v))$;
end
Return the set of cycles S;
while Σ *changes* **do**
  Compute a set S of shortest cycles passing through each edge
  Consider each cycle c in S in increasing order of CV values
  if *c is good and not in* Σ **then**
    Add c to Σ
    Compute the latent coverage level of c
    Update the residual coverage of all cycle nodes, removing nodes with non-positive residual coverage
  else
  end
  end

computing the strongly connected components of and working separately on each one.

2.5 Generating simulated plasmidomes

We simulated error-free paired-end reads from plasmids using BEAR (Johnson *et al.*, 2014), a read simulator designed to generate artificial metagenome data. To avoid introducing coverage drops at sequence ends typical of linear sequences, we modified BEAR (https://github.com/rozovr/BEAR) to allow sampling of reads bridging reference sequence ends, as is observed for circular sequences. Plasmid reference sequences were selected from the NCBI plasmids database and from plasmid sequences reported in (Brown Kav *et al.*, 2013), filtered to include 2760 sequences with a length range of 1–20 kbp with a mean of 6337 bp. Five datasets were created, composed of 100 bp mates (read pair ends), with insert sizes, varying from 1.25 M pairs sampled on 100 reference sequences doubling successively up to 20 M pairs sampled on 1600 sequences. Abundance levels were assigned using BEAR's low complexity option, which concentrates high abundance to few species using a power function with parameters derived from (Pignatelli and Moya, 2011): the function takes the form ci^d, where $c = 31.4$ and $d = -1.28$, and i is iteratively assigned values from 1 to the number of species simulated. These values are then normalized by their sum to yield a probability distribution.

2.6 Evaluating performance

To test recovery of the ground truth sequences by each plasmid detection program, we used the Nucmer alignment tool (Kurtz *et al.*, 2004), which is designed for efficiently comparing long nucleotide sequences such as those of whole plasmids or chromosomes. In order to simplify this process, we modified reference sequences to remove non-ACGT characters before read simulation and alignments. To avoid fragmented alignments caused by differences in start positions, we concatenated each reference sequence to itself before mapping; this allowed identification of complete matches at the center of the concatenated contigs when they were present. Output cycles of each tested program were defined as true positives (TP) if they had 100% identity hits covering at least 80% of one of the reference sequences. False positives (FP) were any output cycles not meeting these criteria, and false negatives (FN) were reference sequences not aligned to in the output set using these criteria. Based on these conventions, precision $= \frac{TP}{TP+FP}$ and recall $= \frac{TP}{TP+FN}$. We used the *F*1 score (Powers, 2011) to combine these measures in a manner that weighs precision and recall equally.

2.7 Primer design and PCR validation of plasmid contigs

The plasmidome dataset was divided into two separate subsets, including simple (single node) cycles ($N = 370$) and complex (multi-node) paths within the graph ($N = 50$). Each of these was divided into coverage bins, and selected representatives from each bin (High coverage: 60–1000x, mid–high coverage: 15–60x, mid-low coverage: 5–15x, low coverage: 1–5x) were validated by PCR. Overall, 24 simple cycles and 39 complex cycles were chosen for PCR validation. From the metagenome dataset ($N = 40$), all assembled plasmids were of the same coverage bin (1–5X) and 10 of them were randomly selected for validation. This was also the case for the *E. coli* E2022 isolate ($N = 4$) for which all plasmids were validated by PCR, aside from a recovered Phi X control sequence. Primers were designed to produce an amplification product only if their template is circular; this was achieved by directing the opposing primers towards the edge of the linear plasmid contig. PCR reactions were carried out using Advantage GC Genomic LA PCR Polymerase (Clontech) according to the manufacturer's instructions. The PCR reactions were as follows: 1.5 μl Advantage buffer (10×), 0.6 μl of each primer (5 mM), 0.15 μl Ex Advantage GC Genomic LA DNA Polymerase, 100 ng of template DNA, 1.5 μl of dNTPs (10 mM) and DDW was added to a final volume of 25 μl. All PCR reactions were carried out in a Sensoquest thermocycler (Gottingen, Germany).

3 Results

We first simulated plasmidomes using known references. We used these data sets to assess Recycler's precision and recall (along with those of alternative methods) by comparing predictions against the ground truth known by the simulation design. We also tested Recycler on real data from two *E. coli* isolates, and both a cow rumen metagenome and plasmidome (Brown Kav *et al.*, 2013). For the bacterial isolates that have been sequenced, predicted plasmids were compared against the reference sequences directly. Since no references are available for metagenome and plasmidome data, we evaluated the accuracy by PCR validation (Jørgensen *et al.*, 2014) and by measuring the proportion of predicted plasmids having proper annotation as done in (Brown Kav *et al.*, 2013). Recycler's inputs were assembly graphs generated by SPAdes version 3.6.2 (Bankevich *et al.*, 2012), and alignments generated by BWA version 0.7.5 (Li and Durbin, 2009).

3.1 Simulated plasmidomes

We simulated paired-end reads from known plasmids, and created five datasets of 100, 200, 400, 8000 and 1600 plasmids. Plasmid abundance was distributed so that few plasmids have high abundance. Dataset sizes were 1.25, 2.5, 5, 10 and 20 M pairs, respectively (see Methods for details). Each such dataset was assembled with SPAdes and subsequently its output contigs and assembly

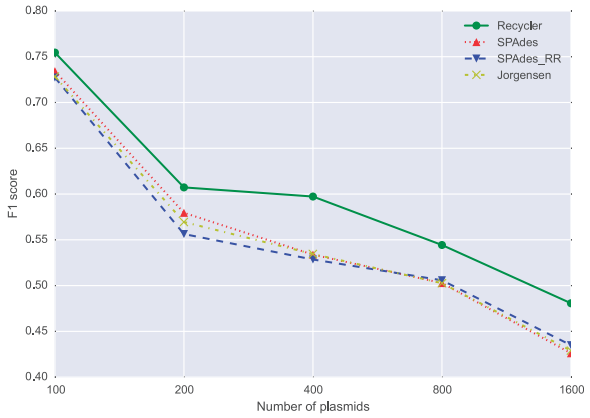

Fig. 2 Methods performance on simulated data. Results are shown for SPAdes without repeat resolution (RR), SPAdes with repeat resolution, the method of Jørgensen *et al.,* and Recycler. The contigs of SPAdes before RR were used as input for the three other methods. Recycler also relied on the graph produced at this stage. F1 score calculation is described in the main text. The *x* axis shows the number of simulated reference sequences in each case

graphs were used as inputs to the tested methods. Recycler was compared with SPAdes with and without repeat resolution (RR), and with a simplified version of Jørgensen's method (described in the Appendix). We used SPAdes' outputs before the repeat resolution stage as inputs to Recycler and to Jørgensen's method, as we found that contigs have greater precision before RR when compared to reference sequences (as shown in Supplementary Table S1). The mapping results are presented in Supplementary Table S1 and Figure 2.

As expected, recall generally decreased as the number of simulated plasmids increased. This was common to all tested methods. In general, we found that Recycler generated more predictions than other methods, leading it to have higher recall than alternative approaches while maintaining high (~90%) precision. The net performance effect is shown in Figure 2 and Supplementary Table S1 in the supplement: Recycler maintains the lead in all cases with 5–14% advantage in both F1 and fraction of true positives. We also found that the number of additional Recycler true positives over those provided by SPAdes generally increased with higher complexity; this culminated in Recycler adding 62 (13%) true positives to SPAdes' output on the 1600 plasmid set (523 versus 461).

To further characterize Recycler's performance, we categorized its predictions in terms of mean total cycle length, number of segments in the cycle (steps), cycle coverage and CV value calculated at the stage the cycle was removed. For each category, values were subdivided into five ranges. In Supplementary Figure S2, we show the precision values and the relative proportions of counts in the specified ranges. Based on this stratification, it can be seen that Recycler shows little dependence on mean coverage or length, but does often preclude candidate cycles that have high CV values or number of steps. This is reflected in the sharp drop-off in the plots as the number of steps or the CV grows.

3.2 Real data

All of Recycler's results on real data were subjected to quantification of annotation results as described in (Brown Kav *et al.*, 2013) and compared against cycles present in the output produced by SPAdes. These results are detailed below and a summary of them can be found in Supplementary Table S2 in the Appendix.

3.2.1 Circular integrity of assembled plasmids

A total of 77 sequences were selected for PCR validation by sampling from the different data types as described in the Section 2.7 above. Overall, 89% of the 77 chosen plasmids were validated by PCR as circular DNA molecules. The predicted plasmids from the different samples did not differ in the success rate of circular validation. As coverage has a key role in *de novo* assembly and Recycler's performance, we wished to measure whether the integrity of assembled plasmids would be affected by varying mean *k*-mer coverage. To this end, we validated circularity of plasmids of different coverage levels ranging from 1x to 1000x divided into bins. As can be seen in Figure 3, there was a slightly lower success rate for the lower coverage plasmids. However, coverage and validation rate were not found to be significantly correlated. Additionally, the high number of predicted plasmids in the plasmidome data set allowed us to measure the effect of the complexity of the path in the graph on the integrity of the plasmids. When more edges are involved in a cycle, it is more complex, and the chance of noise in coverage levels and errors in sequence increases. Thus, we divided this dataset into two bins according to path length on the graph: simple: single node (self-edge) paths, complex: two nodes or more. These two bins did not show difference in their validation rate, further stressing Recycler's strength in extracting plasmids from complex paths.

3.2.2 *E. coli* isolate data

We ran Recycler on two *E. Coli* strains: JJ1886, downloaded from http://www.ebi.ac.uk/ena/data/view/SRX321704, and E2022, sequenced locally. Annotation for plasmids found in both strains was provided in (Lanza *et al.*, 2014); comparisons against Recycler outputs with this annotation are reported in Supplementary Tables S3 and S4. Of the five plasmids known for JJ1886, Recycler output four complete matches (100% identity over 100% length) having lengths 55.9, 5.6, 5.2 and 1.6 kbp. It also output three additional sequences which completely matched previously reported plasmids: two are known to be present in *S. Aureus*, and one in S. Chromogenes. Further tests will be needed in order to validate whether these additional hits are truly present in the sequenced sample, and furthermore, whether they are stable residents of the tested hosts or were present as a result of contamination. When tested on E2022, Recycler performed similarly, recalling most of its known plasmids and outputting a few additional cycles that were complete or near complete matches to known plasmids and one phage. These results are also presented in Supplementary Table S2. In summary, all reported isolate hits represent highly accurate matches to known mobile elements, and most known plasmids for these strains were recovered. In both cases, Recycler missed the longest known reference plasmids; it remains to be seen whether this is due to Recycler's use of a shortest path formulation, lack of significant coverage difference between these plasmids and the host genome, or other factors.

3.2.3 Plasmidome data

A bovine rumen plasmidome sample was prepared as described in (Brown Kav *et al.*, 2013). This data consisted of 5.1 M paired-end 101 bp reads (trimmed to varied sizes for the sake of adapter removal) with an expected insert size of 500 bp [data available upon request]. Recycler output 420 cycles when provided this data. According to ORF prediction performed as in (Brown Kav *et al.*, 2013), 314 of the 420 had significant annotation hits. 96% of those matching annotations either matched plasmid annotations or aligned with plasmids reported in (Jørgensen *et al.*, 2014). Thus, a majority are likely to be plasmids.

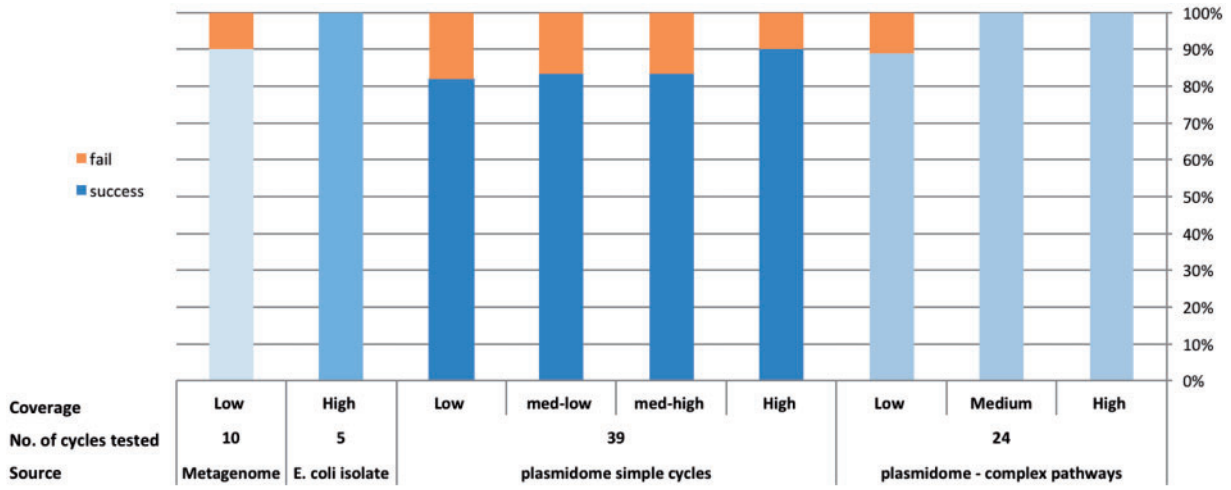

Fig. 3. PCR based validation of Recycler's plasmid predictions. High coverage: 60–1000x, med–high:15–60x, med–low: 5–15x, low: 1–5x

3.2.4 Metagenome data

Metagenome data was derived from the rumen of a different cow residing in the same stable as the cow used to derive the plasmidome data. This data consisted of 7.5 M paired end 150 bp reads with expected insert size of 500 bp [data available upon request]. Recycler produced 40 cycles when run on this data. According to ORF prediction, 37 of the 40 had significant annotation hits. About 35% of those matching annotations either matched plasmid annotations or aligned with plasmids reported in Jørgensen *et al.*, (2014). The proportion of reported cycles matching known plasmid annotations was slightly higher than for simple cycles output by SPAdes (33%). Overall, this test reflects the trend seen elsewhere (Howe *et al.*, 2014) of weak annotation results emerging from metagenome assembly of highly diverse environmental samples.

3.2.5 Comparison with PlasmidSPAdes

Recently, a version of SPAdes tailored for seeking plasmids in isolates, called PlasmidSPAdes, was introduced (Antipov *et al.*, 2016). Unlike Recycler, it does not explicitly seek cycles but removes long edges in the de Bruijn graphs and looks for contigs with coverage significantly different from the mean coverage of the read data. The rationale is that for isolates the coverage distribution is dominated by the host bacterium reads, and the reads of plasmids can be detected as outliers in that distribution. This assumption does not fit plasmidome or metagenome data. PlasmidSPAdes' output is a set of components, each containing a set of contigs with similar mean coverage that putatively originate from the same plasmid. We ran PlasmidSPAdes (packaged with SPAdes 3.80) on the two *E. coli* datasets described above, and compared the results with Recycler's (Supplementary Tables S5 and S6). For E2022, four out of the seven components reported by PlasmidSPAdes matched Recycler's outputs; the shortest of these was among the PCR validated sequences not present in the reference set. Of the three not matching, two seem to have chromosomal origin based on a BLAST search performed on the longest contigs in these components, and the fact that these components had largely tree-like structure: less than half of the component's total length was included in a cycle. Recycler reported one cycle of length 2.1 kb missed by PlasmidSPAdes that was in the reference set. Neither tool succeeded in recovering the longest two plasmids in the reference set.

For JJ1886, three out of the nine components reported matched Recycler's. Of the other six, five likely have chromosomal origin as assessed by the same criteria used for E2022, and one matched a likely plasmid. However, four of these five aligned best with the genome of *S. Aureus*. Recycler reported three additional short sequences between 1.6 and 2.4 kb, each of which had high scoring BLAST hits to plasmids in *S. Aureus* or *S. Chromogenes*. As some of the plasmids reported by both tools also matched *S. Aureus* origin, it is possible that the JJ1886 sample contained a mixture of both cell types. We note that such a mixture could mislead PlasmidSPAdes' estimates of coverage variation, thus allowing large chromosomal fragments to survive filtration.

Overall, aside from the *S. Aureus* sequences observed, the two tools performed similarly on isolate data. This is consistent with the comparison presented in (Antipov *et al.*, 2016). In addition, Recycler can process metagenome and plasmidome graphs, while PlasmidSPAdes can find non-circular plasmids. The two methods primarily differ (when processing isolate data) in what they report for difficult cases involving repeats that are either long or shared by many paths. When Recycler cannot derive a unique circular sequence from a graph component, the component is not included in the output. For PlasmidSPAdes, such components are reported as groups of contigs. In either case, more information (such as long reads) would be needed in order to properly resolve these cases.

4 Discussion

In this article, we describe Recycler, a new algorithm and the first tool available for identification of plasmids from short read-length deep sequencing data. We demonstrate that Recycler discovers plasmids that remain fragmented after *de novo* assembly. We have adapted the approach of choosing among likely enumerated paths using coverage and length properties (often applied in transcriptome assembly (Pertea *et al.*, 2015; Tomescu *et al.*, 2013; Trapnell *et al.*, 2010) for extracting a specific but common inhabitant of metagenomes. We showed that many more real plasmids can be found by only generating likely cycles on the assembly graph versus alternative methods. We validated this approach on both real and simulated data.

Recycler displays high recall and precision on simulated plasmidomes, and we have developed a means of separating real plasmids from cycles due to repeats in isolate data. As we have noted, coverage can be very useful for the latter, but the assumption that coverage will always differ significantly between plasmids and their host

genome does not hold universally. It is worth noting that as new plasmids are identified and their common sequence motifs are observed, both reference-based identification and a priori trained prediction of plasmid features can be improved and harnessed for supplementing identification based on coverage and length features alone. We aim to investigate how such knowledge can be leveraged for increased precision without sacrificing recall.

Furthermore, while Recycler's peeling of lowest CV paths at each step has the advantage of providing a deterministic rule to decide which cycles should be peeled next, this process is heuristic. Better accounting of the uncertainty in observed coverage levels and in the algorithm's dependence on the order of peeling may be obtained by randomizing or repeating parts of the process multiple times. For example, instead of always peeling one best cycle, a random subset of all good cycles may be peeled at once. Repeating this process multiple times and reporting only cycles that persist in a majority of runs may improve both sensitivity and precision.

Further investigation will be needed to assess how plasmids can be extracted from environmental samples, in spite of the limitations now hampering metagenome assembly. This is currently challenging, as diverse genomes require very high coverage for rare species to be captured, but such high coverage data demand computational resources beyond reach of most investigators. While new techniques have aimed to address this problem (Cleary *et al.*, 2015; Howe *et al.*, 2014), they have yet to see widespread use, and work best when paired with multiple samples to allow for species separation by co-abundance signatures. Along with addressing these concerns, it remains to be seen whether a mixed approach of pre-screening environmental samples for plasmids and computationally filtering them out may benefit metagenome graph simplification.

Acknowledgements

RR wishes to thank Kobi Perl and David Pellow for helpful comments given in the preparation of the manuscript.

Funding

This work was supported in part by the Israel Science Foundation [grants no. 1425/13 to EH, 317/13 to RS, and 1313/13 to IM], and the Israel Science Foundation-National Natural Science Foundation of China joint program 2015-18 to RS. Additional support was provided by the European Research Council under the European Union's Horizon 2020 research and innovation program [grant agreement No 640384 to IM] and the Israeli Center of Research Excellence (I-CORE), Gene Regulation in Complex Human Disease, Center No [Grant 41/11 to RS]. RR was supported in part by a fellowship from the Edmond J. Safra Center for Bioinformatics at Tel Aviv University, an IBM PhD fellowship, and by the Center for Absorption in Science, the Israel Ministry of Immigrant Absorption. EH is a Faculty Fellow of the Edmond J. Safra Center for Bioinformatics at Tel Aviv University.

Conflict of Interest: none declared.

References

Antipov,D. *et al.* (2016). plasmidSPAdes: Assembling Plasmids from Whole Genome Sequencing Data. Technical report.

Bankevich,A. *et al.* (2012) SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.*, **19**, 455–477.

Bevan,M.W., Flavell,R.B. and Chilton, M.D. (1983) A chimaeric antibiotic resistance gene as a selectable marker for plant cell transformation. *Nature*, **304**, 184–187.

Brown Kav,A. *et al.* (2012) Insights into the bovine rumen plasmidome. *Proc. Natl. Acad. Sci. USA*, **109**, 5452–5457.

Brown Kav,A. *et al.* (2013) A method for purifying high quality and high yield plasmid DNA for metagenomic and deep sequencing approaches. *J. Microbiol. Methods*, **95**, 272–279.

Cleary,B. *et al.* (2015) Detection of low-abundance bacterial strains in metagenomic datasets by eigengenome partitioning. *Nat. Biotechnol.*, **33**, 1053–1060.

Conlan,S. *et al.* (2014) Single-molecule sequencing to track plasmid diversity of hospital-associated carbapenemase-producing Enterobacteriaceae. *Sci. Translat. Med.*, **6**, 254ra126.

Doring,H. and Starlinger,P. (1984) Barbara McClintock's controlling elements: now at the DNA level. *Cell*, **39**, 253–259.

Gilbert,J.A. and Dupont,C.L. (2011) Microbial metagenomics: beyond the genome. *Annu. Rev. Mar. Sci.*, **3**, 347–371.

Gross,J.L. *et al.* (2013) *Handbook of Graph Theory, 2nd edn*. Chapman & Hall/CRC, Boca Raton, FL.

Halary,S. *et al.* (2009) Network analyses structure genetic diversity in independent genetic worlds. *Proc. Natl. Acad. Sci. USA*, **107**, 127–132.

Hartman,T. *et al.* (2012) How to split a flow? In: *2012 Proceedings IEEE INFOCOM*, pp. 828–836..

Howe,A.C. *et al.* (2014) Tackling soil diversity with the assembly of large, complex metagenomes. *Proc. Natl. Acad. Sci. USA*, **111**, 4904–4909.

Hunt,M. *et al.* (2015) Circlator: automated circularization of genome assemblies using long sequencing reads. *Technical Report*.

Johnson,D.B. (1977) Efficient algorithms for shortest paths in sparse networks. *J. ACM*, **24**, 1–13.

Johnson,S. *et al.* (2014) A better sequence-read simulator program for metagenomics. *BMC Bioinformatics*, **15 Suppl 9**(**Suppl 9**), S14.

Jørgensen,T.S. *et al.* (2014) Hundreds of circular novel plasmids and DNA elements identified in a rat cecum metamobilome. *PLoS One*, **9**, e87924.

Kurtz,S. *et al.* (2004) Versatile and open software for comparing large genomes. *Genome Biol.*, **5**, R12.

Lanza,V.F. *et al.* (2014) Plasmid flux in *Escherichia Coli* ST131 sublineages, analyzed by plasmid constellation network (PLACNET), a new method for plasmid reconstruction from whole genome sequences. *PLoS Genet.*, **10**, e1004766.

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

Neu,H.C. (1992) The crisis in antibiotic resistance. *Science*, **257**, 1064–1073.

Peng,Y. *et al.* (2012) IDBA-UD: a *de novo* assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, **28**, 1420–1428.

Pertea,M. *et al.* (2015) StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nat. Biotechnol.*, **33**, 290–295.

Pignatelli,M. and Moya,A. (2011) Evaluating the fidelity of *de novo* short read metagenomic assembly using simulated data. *PLoS One*, **6**, e19984.

Powers,D.M. (2011). Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation. *J. Mach. Learn. Technol.*, **2**, 37–63.

Prjibelski,A.D. *et al.* (2014) ExSPAnder: a universal repeat resolver for DNA fragment assembly. *Bioinformatics*, **30**, i293–i301.

Tomescu,A.I. *et al.* (2013) A novel min-cost flow method for estimating transcript expression with RNA-Seq. *BMC Bioinformatics*, **14 Suppl 5**, S15.

Trapnell,C. *et al.* (2010) Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nat. Biotechnol.*, **28**, 511–515.

Chapter 4

Faucet: streaming de novo assembly graph construction

Bioinformatics, 2017, 1–8
doi: 10.1093/bioinformatics/btx471
Advance Access Publication Date: 24 July 2017
Original Paper

OXFORD

Genome analysis

Faucet: streaming *de novo* assembly graph construction

Roye Rozov[1], Gil Goldshlager[2], Eran Halperin[3,*] and Ron Shamir[1,*]

[1]Blavatnik School of Computer Science, Tel-Aviv University, Tel Aviv, Israel, [2]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, USA and [3]Departments of Computer Science, Anesthesiology and Perioperative Medicine, University of California Los Angeles, CA, USA

*To whom correspondence should be addressed.

Associate Editor: Cenk Sahinalp

Received on April 12, 2017; revised on July 10, 2017; editorial decision on July 18, 2017; accepted on July 21, 2017

Abstract

Motivation: We present Faucet, a two-pass streaming algorithm for assembly graph construction. Faucet builds an assembly graph incrementally as each read is processed. Thus, reads need not be stored locally, as they can be processed while downloading data and then discarded. We demonstrate this functionality by performing streaming graph assembly of publicly available data, and observe that the ratio of disk use to raw data size decreases as coverage is increased.

Results: Faucet pairs the de Bruijn graph obtained from the reads with additional meta-data derived from them. We show these metadata—coverage counts collected at junction k-mers and connections bridging between junction pairs—contain most salient information needed for assembly, and demonstrate they enable cleaning of metagenome assembly graphs, greatly improving contiguity while maintaining accuracy. We compared Fauceted resource use and assembly quality to state of the art metagenome assemblers, as well as leading resource-efficient genome assemblers. Faucet used orders of magnitude less time and disk space than the specialized metagenome assemblers MetaSPAdes and Megahit, while also improving on their memory use; this broadly matched performance of other assemblers optimizing resource efficiency—namely, Minia and LightAssembler. However, on metagenomes tested, Faucet,o outputs had 14–110% higher mean NGA50 lengths compared with Minia, and 2- to 11-fold higher mean NGA50 lengths compared with LightAssembler, the only other streaming assembler available.

Availability and implementation: Faucet is available at https://github.com/Shamir-Lab/Faucet

Contact: rshamir@tau.ac.il or eranhalperin@gmail.com

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Assembly graphs encode relationships among sequences from a common source: they capture sequences as well as the overlaps observed among them. When assembly graphs are indexed, their sequence contents can be queried without iterating over every sequence in the input. This functionality makes graph and index construction a prerequisite for many applications. Among these are different types of assembly—e.g. *de novo* assembly of whole genomes, transcripts, plasmids etc. (Pertea *et al.*, 2015; Rozov *et al.*, 2017)—and downstream applications—e.g. mapping reads to the graphs, variant calling, pangenome analysis etc. (Iqbal *et al.*, 2012; Novak *et al.*, 2017).

In recent years, much effort has been expended to reduce the amount of memory used for constructing assembly graphs and indexing them. Major advances often relied on index structures that saved memory by enabling subsets of possible queries: e.g. one could query what extensions a given substring s has, but not how many times s was seen in the input data. A great deal of success ensued in reducing the amount of memory needed to efficiently construct the

central data structures used by most *de novo* assembly algorithms, namely, the de Bruijn and string graphs (Chikhi and Rizk, 2012; Pell *et al.*, 2012; Simpson and Durbin, 2010; Ye *et al.*, 2012). Furthermore, efficient conversion of de Bruijn graphs to their *compacted* form (essentially string graphs with fixed overlap size) has been demonstrated Chikhi *et al.*, 2014, 2016; Minkin *et al.*, 2016).

In parallel to these efforts, streaming approaches were demonstrated as alternative resource-efficient means of performing analyses that had typically relied on static indices. Although appealing in terms of speed and low memory use, these approaches were initially demonstrated primarily for counting-centered applications such as estimating k-mer frequencies, error-correction of reads, and quantification of transcripts (Melsted and Halldorsson, 2014; Mohamadi *et al.*, 2017; Roberts and Pachter, 2012; Song *et al.*, 2014; Zhang *et al.*, 2014).

Recently, a first step towards bridging the gap between streaming approaches and those based on static index construction was taken, hinting at the potential benefits of combining the two. El-Metwally *et al.* (2016) demonstrated a streaming approach to assembly by making two passes on a set of reads. The first pass subsamples k-mers in the de Bruijn graph and inserts them into a Bloom filter, and the second uses this Bloom filter to identify 'solid' (likely correct) k-mers, which are then inserted into a second Bloom filter. This streaming approach resulted in very high resource efficiency in terms of memory and disk use. However, LightAssembler finds solid k-mers while disregarding paired-end and coverage information, and thus is limited in its ability to resolve repeats and to differentiate between different possible extensions in order to improve contiguity.

In this work, we extend this approach with the aim of providing a more complete alternative to downloading and storing reads for the sake of *de novo* assembly. We show this is achievable via online graph and index construction. We describe the Faucet algorithm, composed of an online phase and an offline phase. During the online phase, two passes are made on the reads without storing them locally to first load their k-mers into a Bloom filter, and then identify and record structural characteristics of the graph and associated metadata essential for achieving high contiguity in assembly. The offline phase uses all of this information together to iteratively clean and refine the graph structure.

We show that Faucet requires less disk space than the input data, in contrast with extant assemblers that require storing reads and often produce intermediate files that are larger than the input. We also show that the ratio of disk space Faucet uses to the input data improves with higher coverage levels by streaming successively larger subsets of a high coverage human genome sample. Furthermore, we introduce a new cleaning step called *disentanglement* enabled by storage of paired junction extensions in two Bloom filters—one meant for pairings inside a read, and one meant for junctions on separate paired end mates. We show the benefit of disentanglement via extensive experiments. Finally, we compared Faucet's resource use and assembly quality to state of the art metagenome assemblers, as well as leading resource-efficient genome assemblers. Faucet used orders of magnitude less time and disk space than the specialized metagenome assemblers MetaSPAdes and Megahit, while also improving on their memory use; this broadly matched performance of other assemblers optimizing resource efficiency—namely, Minia and LightAssembler. However, on metagenomes tested, Faucet's outputs had 14–110% higher mean NGA50 lengths compared with Minia, and 2- to 11-fold higher mean NGA50 lengths compared with LightAssembler, the only other streaming assembler available.

2 Preliminaries

For a string s, we denote by $s[i]$ the character at position i, $s[i{:}\ j]$ the substring of s from position i to j (inclusive of both ends), and $|s|$ the length of s. Let $pref(s, j)$ be the prefix comprised of the first j characters of s and $suff(s,j)$ be the suffix comprised of the last j characters of s. We denote concatenation of strings s and t by $s^{\circ}t$, and the reverse complement of a string s by s'.

A *k-mer* is a string of length k drawn from the DNA alphabet $\Sigma = \{A, C, G, T\}$. The de Bruijn graph $G(S,k) = (V,E)$ of a set of sequences S has nodes defined by consecutive k-mers in the sequences, $V = \cup_{s\in S} \cup_{i=0}^{|s|-k+1} s[i : i + k - 1]$; E is the set of arcs defined by $(k-1)-mer$ overlaps between nodes in V. Namely, identifying vertices with their k-mers, $(u, v) \in E \iff suff(u, k - 1) = pref(v, k - 1)$. Each node v is identified with its reverse complement v', making the graph G bidirected, in that edges may represent overlaps between either orientation of each node (Medvedev *et al.*, 2007). When necessary, our explicit representation of nodes will use *canonical* node naming, i.e. the name of node (v, v') will be the lexicographically lesser of v and v'. *Junction nodes* are defined as k-mers having in-degree or out-degree > 1. *Terminal nodes* are k-mers having out-degree 1 and in-degree 0 or in-degree 1 and out-degree 0. Terminals and junctions are collectively referred to as *special nodes*. The *compacted de Bruijn graph* is obtained from a de Bruijn graph by merging all adjacent *non-branching nodes* (i.e. those having in-degree and out-degree of exactly 1). The string associated with merged adjacent nodes is the first k-mer, concatenated with the single character extensions of all following non-branching k-mers. Such merged non-branching paths are called *unitigs*.

Since a junction v having in-degree > 1 and out-degree 1 is identified with v' having out-degree > 1 and in-degree 1, we speak of junction directions relative to the reading direction of the junction's k-mer. Therefore, a *forward junction* has out-degree > 1, and a *back junction* has in-degree > 1. We refer to outbound k-mers beginning paths in the direction having out-degree > 1 as *heads*, and the sole outbound k-mer in the opposite direction as the junction's *tail*. It is possible that a junction may have no tail.

A Bloom filter B is a space-efficient probabilistic hash table enabling insertion and approximate membership query operations (Bloom, 1970). The filter consists of a bit array of size m, and an element x is inserted to B by applying h hash functions, $f_0, \ldots, f_{h-1}$ such that $\forall_{i\in[0,h-1]}\ f_i(x) \in\ [0, m - 1]$, and setting values of the filter to 1 at the positions returned. For a Bloom filter B and string s, by $s \in B$ or the term 's in B' we refer to $B[s] = 1$, i.e. when the h hash functions used to load B are applied to s, only 1 values are returned. Similarly, $s \notin B$ or 's not in B' means that at least one of the h hash functions of B returned 0 when applied to s. For any s that has been inserted to B, $B[s] = 1$ by definition (i.e. there are no false negatives). However, false positives are possible, with a probability that can tuned by adjusting m or h appropriately.

3 Materials and methods

We developed an algorithm called Faucet for streaming *de novo* assembly graph construction. A bird's eye view of its entire work-flow is provided in Figure 1. Below we detail individual steps.

3.1 Online Bloom filter loading

Faucet begins by loading two Bloom filters, B_1 and B_2, as it iterates through the reads, using the following procedure: all k-mers are inserted to B_1, and only k-mers already in B_1 (i.e. those for which all hash queries return 1 from B_1) are inserted to B_2. Namely, for each

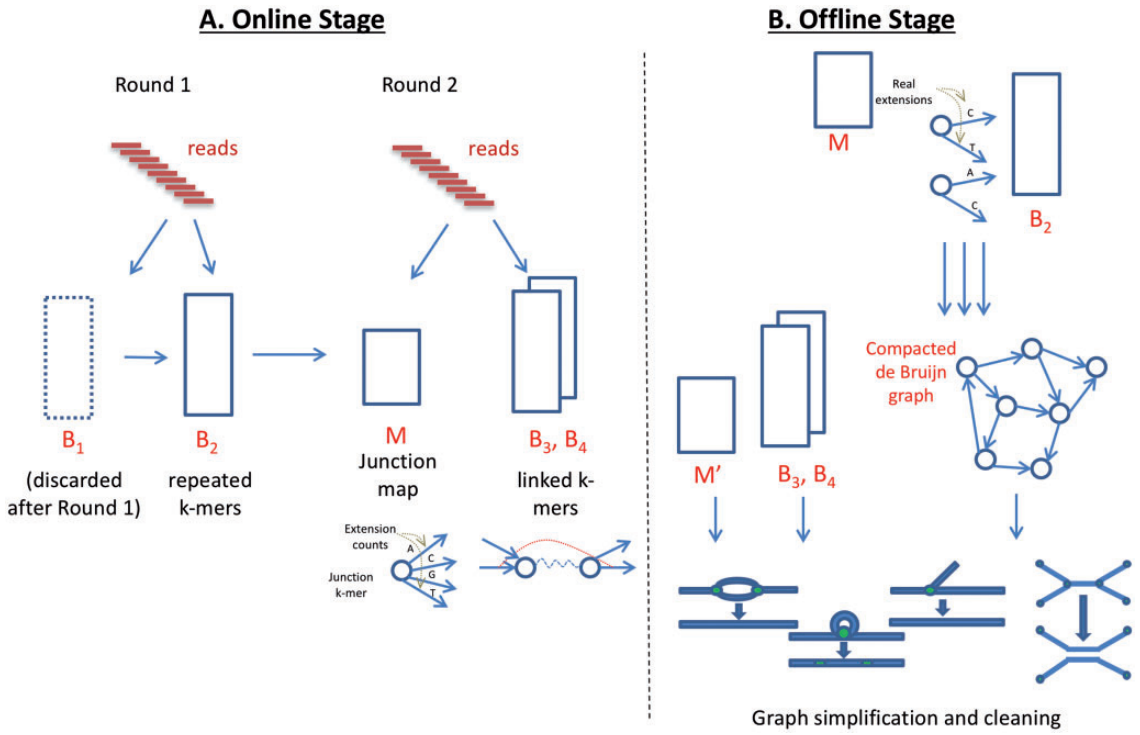

Fig. 1. Faucet work-flow. **(A)** The online stage involves a first round of processing all reads in order to load Bloom filters B_1 and B_2, and a second round in order to build the junction map M and load additional Bloom filters B_3 and B_4. *M* stores the set of all junctions and extension counts for each junction, while B_3 and B_4 capture connections between junction pairs. The two online rounds capture information from and perform processing on each read, and the processing performed always depends on the current state of data structures being loaded. **(B)** The offline stage uses B_2 and *M*, constructed during the online stage, in order to build the compacted de Bruijn graph by extending between special nodes using Bloom filter queries. ContigNodes (not shown) take the place of junctions and are stored in *M′*, allowing access (via stored pointers) to Contigs out of each junction, and coverage information. An additional vector of coverage values at fake or past junctions is also maintained for each Contig. Then, B_3, B_4, and this coverage information are used together to perform simplifications on and cleaning of the graph

k-mer s, if $B_1[s] = 1$ then we insert s into B_2, otherwise we insert into B_1. After iterating through all reads, B_1 is discarded and only B_2 is used for later stages. This procedure imposes a coverage threshold on the vast majority of k-mers so that primarily 'solid k-mers' (Pevzner *et al.*, 2001) observed at least twice are kept. This process is depicted in Round 1 of Figure 1A. We note that a small proportion of singleton or false positive k-mers may evade this filtration. No count information is associated with k-mers at this round.

3.2 Online graph construction

B_2, loaded at the first round, enables Faucet to query possible forward extensions of each k-mer. Faucet iterates through all reads a second time to collect information necessary for avoiding false positive extensions, building the compacted de Bruijn graph, and later, cleaning the graph. The second round consists of finding junctions and terminal k-mers, recording their true extension counts, and recording k-mer pairs (Round 2 of Fig. 1A).

Faucet's Online stage has one main routine—Algorithm 1—that calls upon two subroutines—Algorithms 2 and 3. First, junction k-mers and their start positions are derived from a call to Algorithm 2. To find junctions, Algorithm 2 makes all possible alternate extension queries (Lines 3–5) to B_2 for each k-mer in the read sequence r. A junction k-mer j may have multiple extensions in B_2—either because there are multiple extensions of j in G that are all real (i.e. present on some read), or because there is at least one real extension in G and some others in B_2 that are false positives. Accordingly, each k-mer possessing at least one extension that differs from the next base on the read is identified as a junction. Whenever one is found, its sequence along with its start position are recorded (Line 4), and the list of such tuples is returned. We note that each k-mer in the read is also queried for junctions in the reverse complement direction, but this is not shown in Algorithm 2.

Algorithm 1. *scanReads*(R, B_2)

Input: read set R, Bloom filter B_2 loaded from round 1, an empty Bloom filter B_3
Output: 1. a junction Map M comprised of (*key*, *value*) pairs. Each *key* is a junction k-mer, and each *value* $\in \mathbb{N}^4$ is a vector $[c_A, c_C, c_G, c_T]$ of counts representing the number of times each possible extension of *key* was observed in R; 2. B_3 is loaded with linked k-mer pairs (i.e. specific 2k-mers—see text—are hashed in).

1: $M \leftarrow \varnothing$
2: **for** $r \in R$ **do**
3: $juncs \leftarrow findJunctions(r, B_2)$ ▷ call to Algorithm 2
4: **for** $(junc, pos) \in juncs$ **do**
5: **if** $junc \notin M$ **then**
6: $M[junc] \leftarrow [0, 0, 0, 0]$
7: *increment counter in M for* $r[pos + k]$
$recordPairs(r, juncs, B_3)$ ▷ call to Algorithm 3
8: **return** M,B_3

Algorithm 1 then uses this set of junctions to perform accounting (Lines 4–8). All junctions are inserted into a hash map M that maps junction k-mers to vectors maintaining counts for each extension. For each junction of r, a count of 0 is initialized for each possible extension. These counters are only incremented based on extensions observed on reads—i.e. extensions due to Bloom filter outputs alone are not counted. As every real extension out of each junction must be observed on some read, and we scan the entire set of reads, an extension will have non-zero count only if it is real. This mechanism allows Faucet to maintain coverage counts for all real extensions out of junctions. In later stages, only extensions having non-zero counts will be visited, but counts are stored for real extensions of false junctions as well. These latter counts are used to sample coverage distributions on unitig sequences at more points than just their ends. Proportions of real junctions versus the totals stored after accounting are described in the section 'Solid junction counts' in the Supplementary Appendix.

Algorithm 2. *findJunctions*(r, B_2)

Input: read r and Bloom filter B_2
Output: juncTuples, a list of tuples (seq, p), where p is the start position of junction k-mer seq in r, in order of appearance on r
1: $juncTuples \leftarrow \varnothing$
2: **for** $i \in [0, |r| - k]$ **do** $kmer \leftarrow r[i : i + k - 1]$
3: **for** $c \in \Sigma \backslash \{r[i + k]\}$ **do**
4: **if** $(suff(kmer, k - 1) \circ c \in B_2)$ **then**
$juncTuples \leftarrow juncTuples \cup (kmer, i)$
5: **return** $juncTuples$

Following the accounting performed on observed junctions, Faucet records adjacencies between pairs of junctions using additional Bloom filters—B_3 and B_4. These adjacencies are needed for disentanglement—a cleaning step applied in Faucet's offline stage. Disentanglement, depicted in Figure 2, is a means of repeat resolution. Its purpose is to split paths that have been merged due to the presence of a shared segment—the repeat—in both paths. In order to 'disentangle', or resolve the tangled region into its underlying latent paths, we seek to store sequences that flank opposite ends of the repeat. Pairs of heads observed on reads provide a means of 'reading out' such latent paths by indicating which heads co-occur on sequenced DNA fragments. The application of disentanglement is presented in the section 'Offline graph simplification and cleaning', while we now focus on the mechanism of pair collection and its rationale. To capture short and long range information separately, Bloom filter B_3 holds head pairs on the same read, while B_4 holds heads chosen such that each head is on a different mate of a paired-end read. Algorithm 3 is the process by which pairs are inserted into B_3, and insertion into B_4 is described in the Supplementary Appendix.

In Algorithm 3, we aim to pair heads that are maximally informative. Informative pairs are those that allow us to 'read out' pairs of unitigs that belong to the same latent path. We specifically choose to insert heads because during the offline stage when disentanglement takes place, adjacencies between each unitig starting at an edge to a head and the unitig starting at the edge from the junction to its tail of are known and accessible via pointers to their sequences. Therefore, extension pairs capturing information of direct adjacencies provide no new information. The closest indirect adjacency that may be informative when captured from a read is that between two

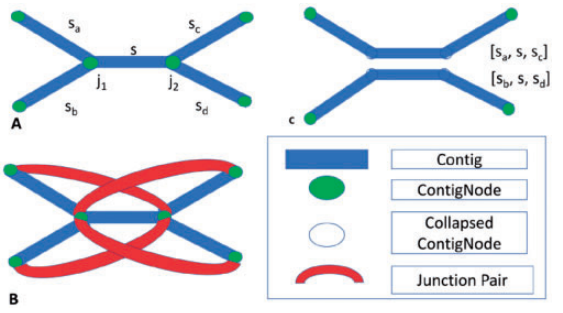

Fig. 2. Disentanglement. **(A)** A *tangle* characterized by two opposite facing junctions j_1 and j_2, each with out-degree 2. **(B)** Junction pairs linking extensions on s_a with s_c and s_b with s_d. Since no pairs link extensions on s_a with s_d or s_b with s_c, only one orientation is supported. **(C)** the result of disentanglement: paths $[s_a,s,s_c]$ and $[s_b,s,s_d]$ are each merged into individual sequences, and junctions j_1 and j_2 are removed from M

junctions that either face in the same direction, or when the first faces back and the second faces forward, as shown in Figure 3A. Thus, when there are only two junctions on a read, their pair of heads is inserted as long as the two junctions are not facing each other. When there are at least three junctions on a read, every other junction out of every consecutive triplet is paired, as shown for a single triplet in Figure 3B. This figure demonstrates that selecting every other head is preferable to selecting consecutive heads out of a triplet. This type of insertion is executed in Lines 1–6 of Algorithm 3 and ensures all unitigs flanking some triplet are potentially inferable. For reads having more than three junctions, applying the triplet rule for every consecutive window of size 3 similarly allows for all unitigs on the read to be included in some hashed pair.

Algorithm 3. *recordPairs*(r, *juncs*, B_3)

Input: read r, *juncs*—a list of pairs (j, p), where p is the start position of junction j in r, and Bloom filter B_3. We also make use of a subroutine $getOutExt(j_i,p_i,r)$ that for a junction j_i returns $pref(j_i, k - 1) \circ r[p_i - k]$ if j_i is a back junction, and $suff(j_i, k - 1) \circ r[p_i + k]$ otherwise.
Output: Bloom filter B_3, loaded with select *linked k-mer pairs*
1: **if** $len(juncs) > 2$ **then**
2: **for** $i \in [0, len(juncs) - 2]$ **do**
3: $back \leftarrow getOutExt(j_i, p_i, r)$
4: $front \leftarrow getOutExt(j_{i+2}, p_{i+2}, r)$
5: $insert(back \circ front, B_3)$ ▷ insert the concatenation into B_3
6: **else if** $(len(juncs) = 2) \wedge (\neg(j_0$ is a forward junction $\wedge$ j_1 is a back junction)) **then**
7: $back \leftarrow getOutExt(j_0, p_0, r)$
8: $front \leftarrow getOutExt(j_1, p_1, r)$
9: $insert(back \circ front, B_3)$
10: **return** B_3

3.3 Offline graph simplification and cleaning

Given B_2, B_3, B_4 and M resulting from the online stage, the compacted de Bruijn graph is generated by traversing each forward extension out of every special k-mer, as well as traversing backwards in the reverse complement direction when the node has not been reached before by a traversal starting from another node. This is done by querying B_2 for extensions and continuing until the next special node is reached. During each such traversal from special node u to special node v, a unitig sequence s_{uv} is constructed. s_{uv} is

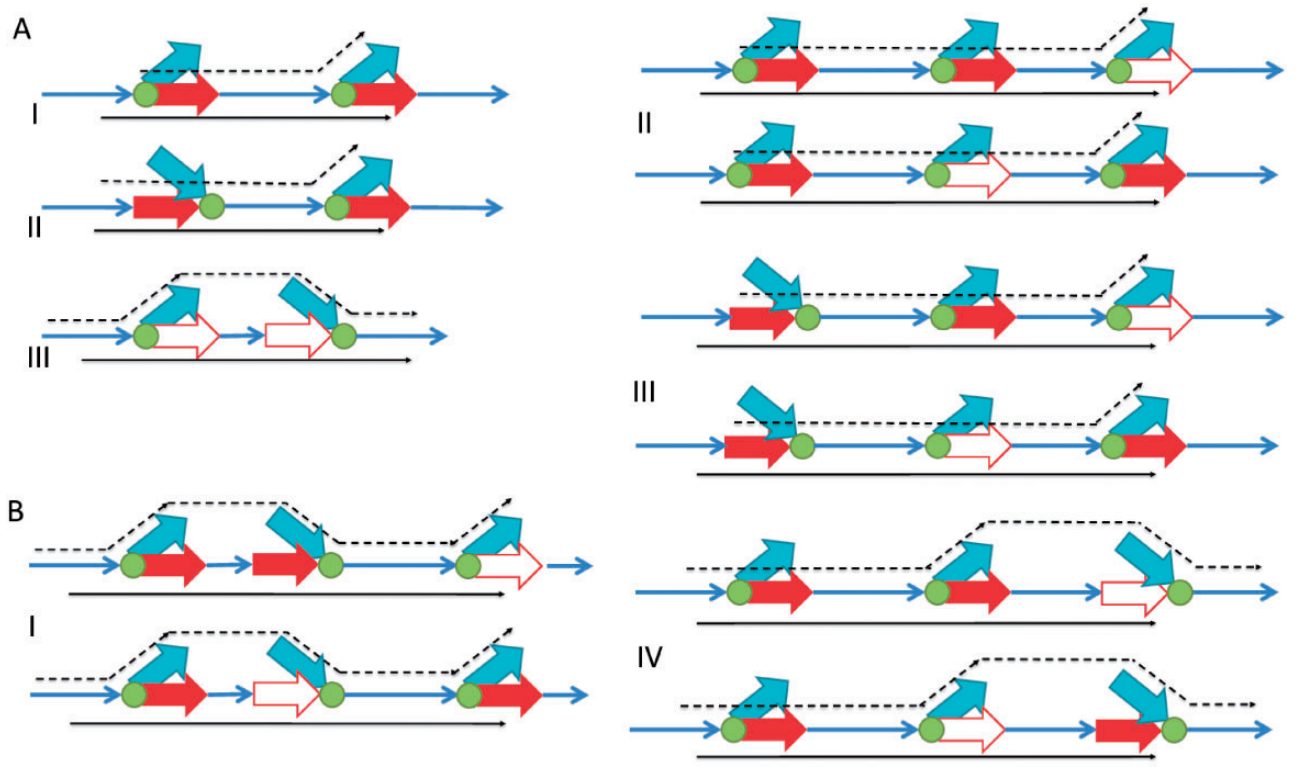

Fig. 3. Rationale for B_3 insertions. Narrow blue arrows indicate unitigs observed on a read, green circles are junctions and thick arrows are junction heads. Among red arrows, solids are those inserted to B_3. For simplicity, we provide a direction to each arrow. The opposite direction is equally valid, hence in this view heads can also enter a junction and not only exit from it. In each case, a pair of red heads is inserted from a read. They will be inserted if they provide additional information to infer a path on the graph. Black lines indicate a subset of possible paths; out of these the solid path is that observed on a read. **(A)** Two junctions observed on a read. I, II: The two heads together imply the solid paths and rule out alternatives, so the pair is inserted to B3. III: The two heads lie on the ends of the same unitig and thus add no information. **(B)** Three junctions observed on a read, comparing insertions of consecutive heads against non-consecutive heads. Four possible arrangements are shown; there are four more that are symmetrical reflections and are not shown to save space. In each case, we compare the unitigs covered (i.e. either having a head on them or being a sole extension at a junctionve heads against non-consecconsecutive (top) and non-consecutive (bottom) junctions are chosen. Note that in Cases I–III the right-most unitig is not covered under consecutive heads

initialized to the sequence of u, and a base is added at each extension until v is reached.

New data structures are constructed in the course of traversals in order to aid later queries and updates. A *ContigNode* structure is used to represent a junction that points to *Contigs*. ContigNodes are structures possessing a pointer to a Contig at each forward extension, as well as one backwards pointer. This backwards pointer connects the junction to the sequence beginning with the reverse complement of the junction's k-mer. Contigs initially store unitig sequences, but these may later be concatenated or duplicated. They also point to one ContigNode at each end. To efficiently query Contigs and ContigNodes, a new hashmap M' is constructed having junction k-mers as keys, and ContigNodes that represent those junctions as values. Isolated contigs formed by unitigs that extend between terminal nodes are stored in a separate set data structure.

Once the raw graph is obtained, cleaning steps commence, incorporating tip removal, chimera removal, collapsing of bulges, and disentanglement. Coverage information and paired-junction links are crucial to these steps. Briefly, tip removal involves deletion of Contigs shorter than the input read length that lead to a terminal node. Chimera and bulge removal steps involve heuristics designed to remove low coverage Contigs when a more credible alternative (higher coverage, or involved in more sub-paths) is identified. These first three steps proceed as described in (Bankevich *et al.*, 2012), thus we omit their full description here.

Disentanglement relies on paired junction links inserted into B_3 and B_4. We iterate through the set of ContigNodes to look for 'tangles'—pairs of opposite-facing junctions joined by a repeat sequence—as shown in Figure 2. Tangles are characterized by tuples (j_1,j_2,s) where j_1 is a back junction, j_2 is a forward junction (or vice-versa), and there is a common Contig s pointed to by the back pointers of both j_1 and j_2. Junctions j_1 and j_2 each have at least two outward extensions. We restrict cleaning to tangles having exactly two extensions at each end. Let s_a and s_b be the Contigs starting at heads of j_1, and s_c and s_d be the Contigs starting at heads of j_2. By disentangling, we seek to pair extensions at each side of s to form two paths. The possible outputs are paths $[s_a,s,s_c]$ together with $[s_b,s,s_d]$ or $[s_a,s,s_d]$ together with $[s_b,s,s_c]$.

Thus, each such pair straddling the tangle—e.g. having one head on s_a and the other on s_c—lends some support to the hypothesis that the correct split is that which pairs the two. To decide between the two possible split orientations, we count the number of pairs supporting each by querying B_3 or B_4 for all possible junction pairings that are separated by a characteristic length associated with the pairs inserted to each. For example, B_3 stores heads out of non-consecutive junction pairs on the same read. Therefore, for each junction on s_a we count each pairing accepted by B_3 with a junction on s_c that is at most one read length away. Specifically for B_3, we also know that inserted pairs are always one or two junctions away from the starting junction, based on the scheme presented in Figure 3. To decide when

a tangle should be split, we apply XOR logic to arrive at a decision: if the count of pairs supporting both paths in one orientation is > 0, and the count of both paths in the other orientation is 0, we disentangle according to the first, as shown in Figure 2. Similar yet more involved reasoning is used for junction links in B_4, using the insert size between read pairs (see Supplementary Appendix). Once we arrive at a decision, we add a new sequence to the set of Contigs that is the concatenation of the sequences involved in the original paths. We note one of the consequences of this simplification step is that the graph no longer represents a de Bruijn graph, in that each k-mer is no longer guaranteed to appear at most once in the graph. Furthermore, the XOR case presented is the most frequently applied form of disentanglement out of a few alternatives. We discuss these alternatives in the Supplementary Appendix.

3.4 Optimizations and technical details

Here we discuss some details omitted from the above descriptions for the sake of completeness. Based on the description of Algorithms 1 and 2, it is possible that false positive extensions out of terminal nodes will ensue. This is possible because the mechanism described for removing false positive junctions can differentiate between one or multiple extensions existing in G for a given node, but cannot differentiate between one or none. This may lead to assembly errors at sink nodes.

To overcome such effects, we store distances between junctions seen on the same read with the distance recorded being assigned to the extension of each junction observed on the read. When an outermost junction on a read has not been previously linked to another junction, we record its distance from the nearest read end—this solves the problem mentioned previously as long as paths to sinks are shorter than read length. To obtain accurate measurements of distances on longer non-branching paths, we also introduce artificial 'dummy' junctions whenever a pre-defined length threshold is surpassed. In effect, this means that reads with no real junctions are assigned dummy junctions.

Once distances and dummy junctions are introduced, an additional benefit is gained: the speed of the read-scan can be improved by skipping between junctions that have been seen before. Once distances are known, if we see a particular extension out of a junction, and then a sequence of length ℓ without any junctions, then, wherever else we see that junction and extension, it must be followed by the exact same ℓ next bases. Otherwise, there would be a junction earlier. So we store ℓ when we see it, and skip subsequent occurrences.

Finally, we note that Faucet can benefit from precise Bloom filter sizing. When a good estimate of dataset parameters is known, the algorithm can do the two-pass process above. Otherwise, to determine the numbers of distinct k-mers and the number of singletons in the dataset in a streaming manner, we have used the tool ntCard (Mohamadi *et al.*, 2017). This requires an additional pass over the reads (for a total of three passes). The added pass does not increase RAM or disk use. In fact, in tests on locally stored data, we found it only adds negligible time.

4 Results

4.1 Assembling while downloading

As a demonstration of streaming assembly, we ran Faucet on publicly available human data, SRR034939, used for benchmarking in (Chikhi and Rizk, 2012). To assess resource use at different data volumes, we ran Faucet on 10, 20 and 37 paired-end files out of 37 total. Streaming was enabled using standard Linux command line tools: wget was used for commencing a download from a supplied URL, and streamed reading from the compressed data was enabled by the bzip2 utility. Downloads were initiated separately for each run. The streaming results are shown in Table 1.

We emphasize that Faucet required less space than the size of the input data in order to assemble it, while most assemblers generate files during the course of their processing that are larger than the input data. Also, the ratio of input data to disk used by Faucet decreased as data volume increased, reflecting the tendency of sequences to be seen repeatedly with high coverage. We also note that Faucet's outputs effectively create a lossy compression of the read data, in that the choice of k value inherently creates some ambiguity for read substrings larger than k. This compression format is also queryable, in that given a k-mer in the graph, its extensions can be found: indeed, this is the basis of Faucet's graph construction and cleaning.

4.2 Disentanglement assessment

To gauge the benefits of disentanglement on assembly quality, we compared Faucet's outputs with and without each of short- and long-range pairing information, provided by Bloom filters B_3 and B_4, on SYN 64—a synthetic metagenome produced to provide a dataset for which the ground truth is known comprised of 64 species (dataset sizes and additional characteristics are provided in the Supplementary Appendix). The results of this assessment are presented in Table 2. We measured assembly contiguity by the NGA50 measure. NG50 is defined as 'the contig length such that using equal or longer length contigs produces x% of the length of the reference genome, rather than x% of the assembly length' in (Gurevich *et al.*, 2013). NGA50 is an adjustment of the NG50 measure designed to penalize contigs composed of misassembled parts by breaking contigs into aligned blocks after alignment to the reference. We found that disentanglement more than doubled contiguity measured by mean NGA50 values, with greater gains as more kinds of disentanglement were enabled. This was also reflected by corresponding gains in the genome fractions, and in the number of species for which at least 50% of the genome was aligned to, allowing NGA50 scores to be reported. More applications of disentanglement also increased the number of misassemblies reported and the duplication ratio; however, two-thirds of the maximum misassembly count is already seen without any disentanglement applied.

Table 1. Resource use and data compression observed as data volume increases

No. of files	Time (h)	RAM (GB)	Disk (GB)	Data size (GB)	Comp. ratio
10	26.3	48.3	19.0	29.6	0.64
20	47.7	84.3	34.3	59.2	0.58
37	98.2	144.7	50.0	108.4	0.46

Table 2. The effect of increasing levels of disentanglement on contiguity and accuracy

Measure	No disent.	B_3 only	B_4 only	both B_3, B_4
Genome fraction (%)	76.4	79.9	80.3	82.3
Dup. ratio	1.00	1.01	1.02	1.02
Mean NGA50	13048	21703	26356	29066
Misassemblies	388	480	521	572
Species reported	54	56	56	56

4.3 Tools comparison

We sought to assess Faucet's effectiveness in assembling metagenomes, and its resource efficiency. For the former, we compared Faucet to MetaSPAdes (Nurk *et al.*, 2017) and Megahit (Li *et al.*, 2014), state of the art metagenome assemblers in terms of contiguity and accuracy that require substantial resources. To address resource efficiency, we also compared Faucet to two leading resource efficient assemblers, Minia 3 (Beta) (Chikhi and Rizk, 2012) and LightAssembler (El-Metwally *et al.*, 2016). We note these last two were not designed as metagenome assemblers, but they perform operations similar to what Faucet does—both in the course of their graph construction steps, and in their cleaning steps. They differ from Faucet in that neither is capable of disentanglement, as they do not utilize paired-end information, but counter this advantage with more sophisticated traversal schemes. All tools were run on two metagenome datasets—SYN64 and HMP—a female tongue dorsum sample sequenced as part of the Human Microbiome Project. Both datasets were used for testing in (Nurk *et al.*, 2017). To achieve a fair comparison, runs were performed with a single thread on the same machine, as Faucet does not currently support multi-threaded execution. Full details of the comparison, including versions, parameters, and data accessions, are presented in the Supplementary Appendix.

Table 3 presents the full results for the tools comparison. There was a strong advantage to Megahit and MetaSPAdes over the three lightweight assemblers (Minia, LightAssembler, and Faucet) in terms of contiguity achieved (shown by NGA50 statistics), but this came at a large cost in terms of memory, disk space, and time, particularly in the case of MetaSPAdes. Among the lightweight assemblers, Minia used by far the most disk space, and differences in other resource measures were less pronounced. Among these three, Faucet had a large advantage in NGA50 statistics relative to the other two. This is highlighted by the trend of Table 3, and shown by its 14–110% advantage in the mean of NGA50 relative to Minia, and 2- to 11-fold advantage relative to LightAssembler.

5 Discussion

Streaming *de novo* assembly presents an opportunity to significantly ease some of the burdens introduced by the recent deluge of second generation sequencing data. We posit the main applications of streaming assembly will be *de novo* assembly of very large individual datasets (e.g. metagenomes from highly diverse environments) and re-assembly of pangenomes derived from many samples. In both cases, very large volumes of data must be digested in order to address the relevant biological questions behind these assays. Therefore, streaming graph assembly presents an attractive alternative to data compression: instead of attempting to reduce the size of data, the aim is to keep locally only relevant information in a manner that is queryable and that allows for future re-analysis.

Here, we have demonstrated a mechanism for performing streaming graph assembly and described some of its characteristics. First, we showed that assembly can be achieved without ever storing raw reads locally. By assembling the graph, an intermediate by-product of many assemblers, we show this technique is generally applicable. By refining the graph and showing better assembly contiguity than competing resource efficient tools on metagenome assembly, we showed this method can also be applied in the setting when sensitive recovery of rare sequences is crucial.

In future work, we aim to expand the capabilities of Faucet in a number of ways. Multi-threaded processing will reduce run times and make the tool more applicable to large data volumes. We believe further refinements of cleaning and contig generation can be achieved by adopting a statistical approach to making assembly decisions. In addition, beyond graph cleaning, we aim to apply Faucet's data structures to path generation, as done with paired end reads in (Nihalani and Aluru, 2016; Prjibelski *et al.*, 2014; Shi *et al.*, 2017). Both have the potential to greatly improve contiguity and accuracy.

Beyond this, this work raises several remaining challenges pertaining to what one may expect of streaming assembly. For instance, it is immediately appealing to ask if streaming assembly can be achieved with a just a single pass on the reads, and if so, what inherent limitations exist. In Song *et al.* (2014), a simple solution is proposed wherein the first 1 M reads are processed to provide a succinct summary for the rest, but such an approach is more suited to high coverage or low entropy data, and thus unlikely to perform well on diverse metagenomes or when rare events are of particular interest. Another issue raised by the performance comparison herein is that of capturing the added value that iterative (multi-k value) graph generation provides. We have given a partial solution by capturing subsets of junction pairs within each read, and between mates of paired-end reads. Although it is possible to iteratively refine the graph with more passes on the reads, each time for the collection of k-mers at different lengths, this becomes unwieldy with large data volumes. Identifying the contexts for which such information would

Table 3. Tool comparison on two metagenomes

	SYN64					HMP				
Measure	Metaspades	Megahit	LightAssembler	Minia	Faucet	Metaspades	Megahit	LightAssembler	Minia	Faucet
Genome fraction (%)	89.1	90.1	75.6	76.5	82.3	46.9	48.6	23.4	27.8	27.9
Duplication ratio	1.02	1.02	1.01	1.00	1.02	1.05	1.12	1.02	1.01	1.05
Mean NGA50 (kb)	167	99.0	2.60	14.6	30.7	28.3	36.8	3.18	6.25	7.12
Median NGA50 (kb)	71.1	57.6	2.30	10.5	23.7	28.3	36.8	3.18	6.25	7.12
Misassemblies	785	949	314	395	572	504	602	100	184	202
Species reported	59	61	55	52	56	12	12	5	3	6
Time (h)	41.2	10.9	1.63	0.97	2.61	30.5	13.0	3.35	0.99	2.30
Memory (GB)	26	9.1	2.7	4.8	6.0	14	8.3	3.4	3.7	7.3
Disk (GB)	43.1	14.3	1.84	28.2	1.59	53.2	11.5	1.30	23.5	1.61

Top values in each cell are for SYN 64 data, and bottom values are for HMP. Duplication ratio is the ratio between the total aligned length to the combined length of all references aligned to. The mean and median NGA50 values are calculated on based on species sufficiently covered by all assemblers to yield an NGA50 value (i.e. 50% of the genome is covered). Species reported are those for which an NGA50 value is reported. In the HMP data, only two species were reported for all, making the mean and median NGA50 values equal. Disk and memory use are those reported by the Linux time utility, and Disk use is the total amount written to disk during the course of a run.

be useful in the graph and indexing the reads to allow for querying of such contexts may provide more efficient means of extracting such information.

Funding

This work was supported in part by the Israel Science Foundation as part of the ISF-NSFC joint program to R.S. R.S. was supported in part by the Raymond and Beverley Chair in Bioinformatics at Tel Aviv University. E.H. was supported in part by the USA–Israel Binational Science Foundation [Grant 2012304] and E.H. and R.R. were supported in part by the Israel Science Foundation [Grant 1425/13]. R.R. was supported in part by a fellowship from the Edmond J. Safra Center for Bioinformatics at Tel Aviv University, an IBM PhD fellowship, and by the Center for Absorption in Science, the Israel Ministry of Immigrant Absorption. E.H. is a Faculty Fellow of the Edmond J. Safra Center for Bioinformatics at Tel Aviv University. G.G. was supported by the MISTI MIT-Israel program at MIT and Tel Aviv University.

Conflict of Interest: none declared.

References

Bankevich,A. *et al*. (2012) SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.*, **19**, 455–477.

Bloom,B.H. (1970) Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, **13**, 422–426.

Chikhi,R. and Rizk,G. (2012) Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms Bioinformatics*, **8**, 236–248.

Chikhi,R. *et al.* (2014). On the representation of de bruijn graphs. In: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. **8394** LNBI, pp 35–55.

Chikhi,R. *et al*. (2016) Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, **32**, i201–i208.

El-Metwally,S. *et al*. (2016) LightAssembler: Fast and memory-efficient assembly algorithm for high-throughput sequencing reads. *Bioinformatics*, **32**, 3215–3223.

Gurevich,A. *et al*. (2013) QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, **29**, 1072–1075.

Iqbal,Z. *et al*. (2012) De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nat. Genet.*, **44**, 226–232.

Li,D. *et al*. (2014) MEGAHIT: An ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics*, **31**, 1674–1676.

Medvedev,P. *et al*. (2007). Computability of models for sequence assembly. In: *Algorithms in Bioinformatics*. Springer, Berlin Heidelberg, pp. 289–301.

Melsted,P. and Halldorsson,B.V. (2014) KmerStream: streaming algorithms for k-mer abundance estimation. *Bioinformatics*, **30**, 3541–3547.

Minkin,I. *et al*. (2016) TwoPaCo: An efficient algorithm to build the compacted de Bruijn graph from many complete genomes. *Bioinformatics*. doi: 10.1093/bioinformatics/btw609.

Mohamadi,H. *et al*. (2017) ntCard: a streaming algorithm for cardinality estimation in genomics data. *Bioinformatics*, **33**, 1324–1330.

Nihalani,R. and Aluru,S. (2016). Effective Utilization of Paired Reads to Improve Length and Accuracy of Contigs in Genome Assembly. In: *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, Seattle, WA, USA — October 02 - 05, 2016, pp. 355–363.

Novak,A.M. *et al*. (2017) Genome graphs. *bioRxiv*. doi: 10.1101/101378.

Nurk,S. *et al*. (2017) metaSPAdes: a new versatile de novo metagenomics assembler. *Genome Res.*, **27**, 824–834.

Pell,J. *et al*. (2012) Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. *Proc. Natl. Acad. Sci. USA*, **109**, 13272–13277.

Pertea,M. *et al*. (2015) StringTie enables improved reconstruction of a transcriptome from RNA-seq reads. *Nat. Biotechnol.*, **33**, 290–295.

Pevzner,P.A. *et al*. (2001) An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci. USA*, **98**, 9748–9753.

Prjibelski,A.D. *et al*. (2014) ExSPAnder: a universal repeat resolver for DNA fragment assembly. *Bioinformatics*, **30**,

Roberts,A. and Pachter,L. (2012) Streaming fragment assignment for real-time analysis of sequencing experiments. *Nat. Methods*, **10**, 71–73.

Rozov,R. *et al*. (2017) Recycler: an algorithm for detecting plasmids from *de novo* assembly graphs. *Bioinformatics*, **33**, 475–482.

Shi,W. *et al*. (2017) The combination of direct and paired link graphs can boost repetitive genome assembly. *Nucleic Acids Res.*, **45**, e43.

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

Song,L. *et al*. (2014) Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biol.*, **15**, 509.

Ye,C. *et al*. (2012) Exploiting sparseness in de novo genome assembly. *BMC Bioinformatics*, **13(Suppl. 6)**, S1.

Zhang,Q. *et al*. (2014) These are not the K-mers you are looking for: Efficient online K-mer counting using a probabilistic data structure. *PLoS One*, **9**, e101271.

Chapter 5

Discussion

In this thesis, we described our work aiming to improve efficiency and scalability of deep sequencing data analysis. We first introduced a new compression algorithm, BARCODE, that allows compression of read sequences relative to a reference much faster than previous approaches. Then, we introduced a new application of de Bruijn assembly graphs for plasmid assembly: the Recycler algorithm generates cycles on the graph, and selects those likely to be plasmids among them. Finally, we introduced Faucet, a new de novo assembler incorporating highly efficient streaming graph construction. In each case, we compared our implementations with a variety of extant methods and showed significant improvements. All algorithms described in this thesis were implemented and made freely available.

In this chapter, we first summarize the methods described in this thesis before characterizing possible extensions of them. Then, we provide broader perspective on the effect likely (based on current trends) technological advances will have on sequence analysis and methods that will be needed to enable it.

5.1 Compressing read sequences

As mentioned in the introduction, short read compression tools are divided into reference-based and reference-free methods. BARCODE, the compression algorithm introduced in Chapter 2, falls between these two camps in that it relies on a reference but eliminates much of the computational burden needed to compress against a reference. Typically, reference based methods require alignment against a reference

in order for reads to be encoded as reference positions along with differences from the reference observed on the read. BARCODE circumvents the need for this alignment step by hashing reads into Bloom filters. BARCODE also avoids the inherent limitations of Bloom filters by carefully accounting for and compressing repetitions of reads, and reads that do not exactly match the reference sequence. This is made possible by employing a trick previously used in de novo assembly [77] - anticipating the set of queries that will be made on a Bloom filter for some static set, and recording the set of false positives to later avoid them.

BARCODE was demonstrated to compress much faster than reference based methods, and achieve better compression ratios than reference-free methods. This is made possible via optimizing the use of Bloom filters to encode both reads matching the reference and false positives efficiently. Further such optimization may be possible, i.e., encoding read multiplicities or avoiding false positive reads by inserting them to additional Bloom filters. However, these optimizations must be weighed against the costs they introduce: additional insertions introduce a penalty in run time, and filtering false positives by checking for multiple insertions may increase the quantity of false negatives that must be encoded.

5.2 Assembling plasmids

Plasmids are responsible for horizontal gene transfer between microbes and can confer advantages to microbes in specific environments [88]. They have been implicated in conferring resistance to antibiotics [89], leading to heightened interest in their study. Up until recently, assembly of plasmids was limited to checking for signs of circularity among contigs output by available assemblers [90], or identifying plasmid contigs by the presence of genes matching plasmid annotation [91]. The Recycler algorithm described in Chapter 3 introduced a simple but much more effective approach: Recycler identifies cycles that belong to plasmids in the assembly graph output by off-the-shelf assemblers.

We initially tested an ILP formulation aimed to assign coverage levels to candidate cycles with the objective being to minimize the sum of differences relative to observed coverage levels of all contigs. Naturally, we found this approach sensitive to choice of candidate cycle set. Also, this approach is more apt for plasmidome samples, but less so for metagenome and isolate samples, where the majority of

contigs usually are not plasmid origin. These issues were partially involved in our moving to a 'progressive' peeling approach. Using minimal weight cycles provides a simple cycle proposal mechanism and was applicable on all desired data types.

Recycler begins by generating a set of candidate cycles for each connected component in the graph. It then selects among all candidates by ranking candidates according to the variance in coverage of contigs in the cycle relative to the mean of their coverage. Recycler was shown to be more sensitive and precise than competing methods, based on assessments via simulated and real data. Furthermore, among predicted plasmids tested via PCR, nearly 90% were validated.

5.3 Streaming assembly

Similar to BARCODE in Chapter 2, Faucet is designed to bridge a gap between two classes of tools. In this case, Faucet is designed to be highly resource efficient while assembling much better than extant resource-frugal approaches by using information typically limited to more heavy-weight assemblers. The main novelty of Faucet is its streaming approach to graph construction that allows this additional information to be captured and efficiently encoded. We showed this streaming approach effectively compresses input data, and that this compression becomes more efficient as read coverage increases. The graph encoding Faucet constructs is queryable for the sake of assembly, introducing a new means of queryable compression that may find additional applications. In comparison with extant methods, Faucet generated much more contiguous assemblies than the lightweight assemblers minia [77] and LightAssembler [92], while being faster and memory efficient than the best available metagenome assemblers, metaSPAdes [93] and MegaHit [94]. Faucet was shown to be comparable in terms of disk use to the only other streaming assembler, LightAssembler [92], but achieved much higher contiguity of assemblies.

While Faucet is more efficient than MegaHit and MetaSPAdes in resource use, these assemblers generate larger contigs. As mentioned in the discussion of Chapter 4, we believe further refinements of cleaning and contig generation can be achieved by adopting a statistical approach to making assembly decisions. In addition, beyond graph cleaning, we aim to apply Faucet's data structures to path generation, as done with paired end reads in [95, 96, 97]. Both have the potential to greatly improve contiguity and accuracy.

5.4 Future research and developments

The landscape of sequencing is changing rapidly, with emergence of new technologies leading to sudden tectonic shifts. Although the time frame is difficult to predict, it is likely that sequencers will eventually be 'chromosome readers' - sequencing full length DNA molecules with very high accuracy at negligible cost. This will change the emphasis of future analysis tasks from de novo assembly to identification by alignment, but a long road of development needs to be traversed in order to enable this change. We describe immediate first steps motivated by the work in this thesis, and a broader discussion of current and future developments.

5.4.1 Storage of read data

Given sequence growth trends, it will be essential to move from storage of read data to maximally informative, minimally redundant representations of sequences. Ideally, these would be full representations of source molecules, such as full haplotypes of human chromosomes or full microbial genomes without gaps or errors. Variation graphs will be needed to non-redundantly encode sequences of related entities or those from the same population [45]. Such graphs will serve as a bridge between the current length limitations and biases of short reads and the high error rates of long reads to leverage the advantages of each until both are improved.

Currently, no methods that we know of have been proposed for compression of single molecule reads. This is likely due to the fact that these reads are inherently harder to compress because of their high error rate, and that one of the most efficient lossy compressions of them is their assembly. The process of assembly requires removal of likely errors and thus eliminates most features that are resistant to compression. Also, the assembled result is much more useful for downstream analysis, as long as little information present in the reads is lost. Recent methods have evolved to retain nearly all implied path information present in individual short reads [98, 99], including Chapter 4 of this thesis. However, currently no scalable solution for retaining full paired end information is known.

Along with retention of information salient for downstream applications, enabling efficient querying of large sequence databases will prove essential. Recent studies that focused on this have greatly increased the scale of data that may be analyzed

simultaneously; more progress will be essential to enable analysis of data corpora, even when they are publicly available [18, 20, 19].

5.4.2 Optimizing construction and refinement of assembly graphs

Assembly graph construction algorithms will need to continue to advance in order to keep pace with data production. Until databases are complete enough to allow identification of most sequenced fragments via alignment against them, there will be a drive to fill in gaps in the tree of life via environmental sequencing, as has been recently seen for viruses [100] and soil bacteria [101], and to fully characterize variation inside populations. Once such studies become prevalent, graph construction will need to be done on streamed data, and graphs will need to be stored in a manner that allows them to be dynamically updated.

To enhance scalability, a parallelized single pass streaming approach would be attractive for coping with very large metagenome or population graphs. If the k-mer loading and junction labeling steps of Faucet are combined, a single pass approach can be obtained, but it remains to be seen what limitations this imposes on graph construction in terms of accuracy and contiguity that can be achieved. Currently, both parallelized and single pass processing introduce challenges in discovery of junctions.

An alternative to streaming data is partitioning it and processing it in a distributed fashion. In this way, memory use at each node is kept manageable, and run times are reduced. Various modes of parallelization have been tested for de Bruijn graph assembly, such as distribution to thousands of cores on supercomputers [102, 103], and distribution to a cluster under the MapReduce framework, implemented in Spark [104]. Since the latter technology is more readily available and allows leveraging cloud infrastructure to access more nodes as needed, it has greater potential for wide adoption. Unfortunately, the implementation described in [104] is not publicly available.

5.4.3 New means of assembly graph simplification, repeat resolution

Until perfect complete molecule sequencing is achieved, algorithms for assembly graph simplification will be needed in order to resolve repeats and differentiate between errors and true variants. Work has been done on defining the information-theoretic thresholds on sequence lengths and error rates at which full genome reconstruction becomes achievable [105, 106]. Recent extensions of this work by some of the same authors apply similar analysis to defining resolution limits when data is not sufficient for perfect recovery [107] and when coverage information may be essential to separate between molecules sharing a repeat, such as in metagenome or transcript assembly [108].

Such studies inform algorithm development and experimental design. While long reads have made complete bacterial genomes routinely achievable, this has yet to be the case for larger-scale applications. For metagenome assembly, it is possible that much greater coverage than has been employed thus far is needed for complete recovery. One of the implications of this being the case would be that more than a Tb of data would be needed in order to fully sequence highly diverse metagenomes.

For plasmid assembly, integration of long reads has proven very useful in recovery of isolate genome plasmids. However, as both plasmidSPAdes and Recycler rely on coverage information to identify likely plasmids, it is likely that the combination of short reads providing a coverage signal and long reads allowing bridging of repeats will prove most useful, particularly in recovery of plasmids out of metagenomes. Further improvements in Recycler's processing may be achieved by refinement of the candidate cycle generation procedure to allow longer cycles, bootstrapping the choice of cycles peeled to allow greater sensitivity, and implementation of dynamic updates on shortest path calculations to improve efficiency and scalability. An interesting application of plasmid assembly may be refinement of metagenome assemblies: each time cycles due to plasmids are peeled off of a metagenome assembly graph, the graph is simplified, leading to longer contigs and simplified graph structure, as long as true plasmids are identified.

Bibliography

[1] Christina Farr. Illumina, Secret Giant Of DNA Sequencing, Is Bringing Its Tech To The Masses. https://www.fastcompany.com/3061591/illumina-owns-the-dna-sequencing-market-now-its-building-an-app-store-too, 2016.

[2] Illumina Inc. History of Illumina Sequencing. https://www.illumina.com/science/technology/next-generation-sequencing/illumina-sequencing-history.html, 2017.

[3] Lex Nederbragt. Developments in NGS. 2016.

[4] John Michael Gaziano, John Concato, Mary Brophy, Louis Fiore, Saiju Pyarajan, James Breeling, Stacey Whitbourne, Jennifer Deen, Colleen Shannon, Donald Humphries, Peter Guarino, Mihaela Aslan, Daniel Anderson, Rene LaFleur, Timothy Hammond, Kendra Schaa, Jennifer Moser, Grant Huang, Sumitra Muralidhar, Ronald Przygodzki, and Timothy J. O'Leary. Million Veteran Program: A mega-biobank to study genetic influences on health and disease. *Journal of Clinical Epidemiology*, 70:214–223, 2016.

[5] Lior Pachter. *Seq. https://liorpachter.wordpress.com/seq/, 2013.

[6] J. Dekker, Karsten Rippe, Martijn Dekker, and Nancy Kleckner. Capturing Chromosome Conformation. *Science*, 295(5558):1306–1311, 2002.

[7] Nicholas J Loman. Thar she blows! Ultra long read method for nanopore sequencing · Loman Labs. http://lab.loman.net/2017/03/09/ultrareads-for-nanopore/.

[8] Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P Drake, Jane M Landolin, and Adam M Phillippy. Assembling large genomes with single-

molecule sequencing and locality-sensitive hashing. *Nature Biotechnology*, 33(6):623–630, 2015.

[9] Shi-Yi Chen, Feilong Deng, Xianbo Jia, Cao Li, and Song-Jia Lai. A transcriptome atlas of rabbit revealed by PacBio single-molecule long-read sequencing. *Scientific Reports*, 7(1):7648, 2017.

[10] Jun Li, Yuka Harata-Lee, Matthew D Denton, Qianjin Feng, Judith R Rathjen, Zhipeng Qu, and David L Adelson. Long read reference genome-free reconstruction of a full-length transcriptome from Astragalus membranaceus reveals transcript variants involved in bioactive compound biosynthesis. *Cell discovery*, 3:17031, 2017.

[11] Joshua Quick, Nicholas J. Loman, Sophie Duraffour, Jared T. Simpson, Ettore Severi, Lauren Cowley, Joseph Akoi Bore, Raymond Koundouno, Gytis Dudas, Amy Mikhail, Nobila Ouédraogo, Babak Afrough, Amadou Bah, Jonathan H. J. Baum, Beate Becker-Ziaja, Jan Peter Boettcher, Mar Cabeza-Cabrerizo, Álvaro Camino-Sánchez, Lisa L. Carter, Juliane Doerrbecker, Theresa Enkirch, Isabel García Dorival, Nicole Hetzelt, Julia Hinzmann, Tobias Holm, Liana Eleni Kafetzopoulou, Michel Koropogui, Abigael Kosgey, Eeva Kuisma, Christopher H. Logue, Antonio Mazzarelli, Sarah Meisel, Marc Mertens, Janine Michel, Didier Ngabo, Katja Nitzsche, Elisa Pallasch, Livia Victoria Patrono, Jasmine Portmann, Johanna Gabriella Repits, Natasha Y. Rickett, Andreas Sachse, Katrin Singethan, Inês Vitoriano, Rahel L. Yemanaberhan, Elsa G. Zekeng, Trina Racine, Alexander Bello, Amadou Alpha Sall, Ousmane Faye, Oumar Faye, N'Faly Magassouba, Cecelia V. Williams, Victoria Amburgey, Linda Winona, Emily Davis, Jon Gerlach, Frank Washington, Vanessa Monteil, Marine Jourdain, Marion Bererd, Alimou Camara, Hermann Somlare, Abdoulaye Camara, Marianne Gerard, Guillaume Bado, Bernard Baillet, Déborah Delaune, Koumpingnin Yacouba Nebie, Abdoulaye Diarra, Yacouba Savane, Raymond Bernard Pallawo, Giovanna Jaramillo Gutierrez, Natacha Milhano, Isabelle Roger, Christopher J. Williams, Facinet Yattara, Kuiama Lewandowski, James Taylor, Phillip Rachwal, Daniel J. Turner, Georgios Pollakis, Julian A. Hiscox, David A. Matthews, Matthew K. O' Shea, Andrew McD. Johnston, Duncan Wilson, Emma Hutley, Erasmus Smit, Antonino Di Caro, Roman Wölfel, Kilian Stoecker, Erna Fleischmann, Martin Gabriel, Simon A. Weller, Lamine Koivogui, Boubacar Di-

allo, Sakoba Keïta, Andrew Rambaut, Pierre Formenty, Stephan Günther, and Miles W. Carroll. Real-time, portable genome sequencing for Ebola surveillance. *Nature*, 530(7589):228–232, 2016.

[12] Alexander L. Greninger, Samia N. Naccache, Scot Federman, Guixia Yu, Placide Mbala, Vanessa Bres, Doug Stryke, Jerome Bouquet, Sneha Somasekar, Jeffrey M. Linnen, Roger Dodd, Prime Mulembakani, Bradley S. Schneider, Jean-Jacques Muyembe-Tamfum, Susan L. Stramer, and Charles Y. Chiu. Rapid metagenomic identification of viral pathogens in clinical samples by real-time nanopore sequencing analysis. *Genome Medicine*, 7(1):99, 2015.

[13] T F Smith and M S Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–7, 1981.

[14] S B Needleman and C D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–53, 1970.

[15] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

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

[17] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

[18] Brad Solomon and Carl Kingsford. Fast search of thousands of short-read sequencing experiments. *Nature Biotechnology*, 34(3):300–302, 2016.

[19] Victoria Popic, Volodymyr Kuleshov, Michael Snyder, and Serafim Batzoglou. GATTACA: Lightweight Metagenomic Binning With Compact Indexing Of Kmer Counts And MinHash-based Panel Selection. *bioRxiv*, 2017.

[20] Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman, Sergey Koren, and Adam M. Phillippy. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17(1):132, 2016.

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

[22] Gene Myers. Efficient Local Alignment Discovery amongst Noisy Long Reads. pages 52–67. Springer, Berlin, Heidelberg, 2014.

[23] Stefan Canzar and Steven L. Salzberg. Short Read Mapping: An Algorithmic Tour. *Proceedings of the IEEE*, 105(3):436–458, 2017.

[24] Melanie Schirmer, Rosalinda D'Amore, Umer Z. Ijaz, Neil Hall, and Christopher Quince. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. *BMC Bioinformatics*, 17(1):125, 2016.

[25] Sara Goodwin, John D. McPherson, and W. Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, 17(6):333–351, 2016.

[26] Yuval Benjamini and Terence P. Speed. Summarizing and correcting the GC content bias in high-throughput sequencing. *Nucleic Acids Research*, 40(10):e72–e72, 2012.

[27] John Eid, Adrian Fehr, Jeremy Gray, Khai Luong, John Lyle, Geoff Otto, Paul Peluso, David Rank, Primo Baybayan, Brad Bettman, Arkadiusz Bibillo, Keith Bjornson, Bidhan Chaudhuri, Frederick Christians, Ronald Cicero, Sonya Clark, Ravindra Dalal, Alex DeWinter, John Dixon, Mathieu Foquet, Alfred Gaertner, Paul Hardenbol, Cheryl Heiner, Kevin Hester, David Holden, Gregory Kearns, Xiangxu Kong, Ronald Kuse, Yves Lacroix, Steven Lin, Paul Lundquist, Congcong Ma, Patrick Marks, Mark Maxham, Devon Murphy, Insil Park, Thang Pham, Michael Phillips, Joy Roy, Robert Sebra, Gene Shen, Jon Sorenson, Austin Tomaney, Kevin Travers, Mark Trulson, John Vieceli, Jeffrey Wegener, Dawn Wu, Alicia Yang, Denis Zaccarin, Peter Zhao, Frank Zhong, Jonas Korlach, and Stephen Turner. Real-Time DNA Sequencing from Single Polymerase Molecules. *Science*, 323(5910), 2009.

[28] Seung Chul Shin, Do Hwan Ahn, Su Jin Kim, Hyoungseok Lee, Tae-Jin Oh, Jong Eun Lee, and Hyun Park. Advantages of Single-Molecule Real-Time Sequencing in High-GC Content Genomes. *PLoS ONE*, 8(7):e68824, 2013.

[29] Erick W Loomis, John S Eid, Paul Peluso, Jun Yin, Luke Hickey, David Rank, Sarah McCalmon, Randi J Hagerman, Flora Tassone, and Paul J Hagerman. Sequencing the unsequenceable: expanded CGG-repeat alleles of the fragile X gene. *Genome Research*, 23(1):121–8, 2013.

[30] Miten Jain, Sergey Koren, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, Sunir Malla, Hannah Marriott, Karen H Miga, Tom Nieto, Justin O'Grady, Hugh E Olsen, Brent S Pedersen, Arang Rhie, Hollian Richardson, Aaron Quinlan, Terrance P Snutch, Louise Tee, Benedict Paten, Adam M. Phillippy, Jared T Simpson, Nicholas James Loman, and Matthew Loose. Nanopore sequencing and assembly of a human genome with ultra-long reads. *bioRxiv*, 2017.

[31] Camilla L.C. Ip, Matthew Loose, John R. Tyson, Mariateresa de Cesare, Bonnie L. Brown, Miten Jain, Richard M. Leggett, David A. Eccles, Vadim Zalunin, John M. Urban, Paolo Piazza, Rory J. Bowden, Benedict Paten, Solomon Mwaigwisya, Elizabeth M. Batty, Jared T. Simpson, Terrance P. Snutch, Ewan Birney, David Buck, Sara Goodwin, Hans J. Jansen, Justin O'Grady, Hugh E. Olsen, MinION Analysis Consortium, and Reference. MinION Analysis and Reference Consortium: Phase 1 data release and analysis. *F1000Research*, 4, 2015.

[32] Jared T Simpson, Rachael E Workman, P C Zuzarte, Matei David, L J Dursi, and Winston Timp. Detecting DNA cytosine methylation using nanopore sequencing. *Nature Methods*, 14(4):407–410, 2017.

[33] Keith Robison. Omics! Omics!: Catching Up On Oxford Nanopore News: More, Better, Meth & Huge. http://omicsomics.blogspot.co.il/2017/03/catching-up-on-oxford-nanopore-news.html.

[34] Zachary D. Stephens, Skylar Y. Lee, Faraz Faghri, Roy H. Campbell, Chengxiang Zhai, Miles J. Efron, Ravishankar Iyer, Michael C. Schatz, Saurabh Sinha, and Gene E. Robinson. Big Data: Astronomical or Genomical? *PLOS Biology*, 13(7):e1002195, 2015.

[35] National Human Genome Research Institute. 2001 Release: First Analysis of Human Genome - National Human Genome Research Institute

(NHGRI). https://www.genome.gov/10002192/2001-release-first-analysis-of-human-genome/.

[36] Po-Ru Loh, Michael Baym, and Bonnie Berger. Compressive genomics. *Nature Biotechnology*, 30(7):627–630, 2012.

[37] Dan. Gusfield. *Algorithms on strings, trees, and sequences : computer science and computational biology*. Cambridge University Press, 1997.

[38] M. Burrows, M. Burrows, and D. J. Wheeler. A block-sorting lossless data compression algorithm. *SRC Scientific Reports*, 1994.

[39] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.

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

[41] Andrei Broder and Michael Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1:636—-646, 2002.

[42] W James Kent. BLAT–the BLAST-like alignment tool. *Genome Research*, 12(4):656–64, 2002.

[43] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. Versatile and open software for comparing large genomes. *Genome Biology*, 5(2):R12, 2004.

[44] Niranjan Nagarajan and Mihai Pop. Sequence assembly demystified. *Nature Reviews Genetics*, 14(3):157–167, 2013.

[45] Adam M Novak, Glenn Hickey, Erik Garrison, Sean Blum, Abram Connelly, Alexander Dilthey, Jordan Eizenga, M. A. Saleh Elmohamed, Sally Guthrie, André Kahles, Stephen Keenan, Jerome Kelleher, Deniz Kural, Heng Li, Michael F Lin, Karen Miga, Nancy Ouyang, Goran Rakocevic, Maciek Smuga-Otto, Alexander Wait Zaranek, Richard Durbin, Gil McVean, David Haussler, and Benedict Paten. Genome Graphs. *bioRxiv*, 2017.

[46] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 44(2):226–232, 2012.

[47] Markus J. Bauer, Anthony J. Cox, and Giovanna Rosone. Lightweight BWT Construction for Very Large String Collections. pages 219–231. Springer, Berlin, Heidelberg, 2011.

[48] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398. IEEE Comput. Soc.

[49] Saul Schleimer, Daniel S. Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03*, page 76, New York, New York, USA, 2003. ACM Press.

[50] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount, and J. A. Yorke. Reducing storage requirements for biological sequence comparison. *Bioinformatics*, 20(18):3363–3369, 2004.

[51] Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, and Carl Kingsford. Compact Universal k-mer Hitting Sets. pages 257–268. Springer, Cham, 2016.

[52] Guillaume Marçais, David Pellow, Daniel Bork, Yaron Orenstein, Ron Shamir, and Carl Kingsford. Improving the performance of minimizers and winnowing schemes. *Bioinformatics*, 33(14):i110–i117, 2017.

[53] Andrei Z. Broder. On the Resemblance and Containment of Documents. *In Compression and Complexity of Sequences (SEQUENCES'97*, pages 21—-29, 1997.

[54] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, pages 20–29, New York, New York, USA, 1996. ACM Press.

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

[56] Páll Melsted and Jonathan K Pritchard. Efficient counting of k-mers in DNA sequences using a bloom filter. *BMC Bioinformatics*, 12(1):333, 2011.

[57] Guillaume Rizk, Dominique Lavenier, and Rayan Chikhi. DSK: k-mer counting with very low memory usage. *Bioinformatics*, 29(5):652–3, 2013.

[58] Sebastian Deorowicz, Agnieszka Debudaj-Grabysz, and Szymon Grabowski. Disk-based k-mer counting on a PC. *BMC Bioinformatics*, 14(1):160, 2013.

[59] Qingpeng Zhang, Jason Pell, Rosangela Canino-Koning, Adina Chuang Howe, and C. Titus Brown. These Are Not the K-mers You Are Looking For: Efficient Online K-mer Counting Using a Probabilistic Data Structure. *PLoS ONE*, 9(7):e101271, 2014.

[60] Qingpeng Zhang, Sherine Awad, and C. Titus Brown. Crossing the streams: a framework for streaming analysis of short DNA sequencing reads. 2015.

[61] L Song, L Florea, and B Langmead. Lighter: fast and memory-efficient sequencing error correction without counting. *Genome Biology*, 2014.

[62] Deniz Yorukoglu, Yun William Yu, Jian Peng, and Bonnie Berger. Compressive mapping for next-generation sequencing. *Nature Biotechnology*, 34(4):374–376, 2016.

[63] Ernest Turro, Shu-Yi Su, Ângela Gonçalves, Lachlan JM Coin, Sylvia Richardson, and Alex Lewin. Haplotype and isoform specific expression estimation using multi-mapping RNA-seq reads. *Genome Biology*, 12(2):R13, 2011.

[64] Nicolas L Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. *Nature Biotechnology*, 34(5):525–527, 2016.

[65] Rob Patro, Geet Duggal, Michael I Love, Rafael A Irizarry, and Carl Kingsford. Salmon provides fast and bias-aware quantification of transcript expression. *Nature Methods*, 14(4):417–419, 2017.

[66] L. Schaeffer, H. Pimentel, N. Bray, P. Melsted, and L. Pachter. Pseudoalignment for metagenomic read assignment. *Bioinformatics*, 33(14):2082–2088, 2017.

[67] James K Bonfield and Matthew V Mahoney. Compression of FASTQ and SAM format sequencing data. *PloS one*, 8(3):e59190, 2013.

[68] Christos Kozanitis, Chris Saunders, Semyon Kruglyak, Vineet Bafna, and George Varghese. Compressing genomic sequence fragments using SlimGene. *Journal of computational biology : a journal of computational molecular cell biology*, 18(3):401–13, 2011.

[69] Faraz Hach, Ibrahim Numanagic, and S Cenk Sahinalp. DeeZ: reference-based compression by local assembly. *Nature Methods*, 11(11):1082–1084, 2014.

[70] Faraz Hach, Ibrahim Numanagic, Can Alkan, and S Cenk Sahinalp. SCALCE: boosting sequence compression algorithms using locally consistent encoding. *Bioinformatics*, 28(23):3051–7, 2012.

[71] Rob Patro and Carl Kingsford. Data-dependent bucketing improves reference-free compression of sequencing reads. *Bioinformatics*, 31(17):2770–2777, 2015.

[72] Szymon Grabowski, Sebastian Deorowicz, and Lukasz Roguski. Disk-based compression of data from genome sequencing. *Bioinformatics*, 31(9):1389–1395, 2015.

[73] Daniel C Jones, Walter L Ruzzo, Xinxia Peng, and Michael G Katze. Compression of next-generation sequencing reads aided by highly efficient de novo assembly. *Nucleic Acids Research*, 40(22):e171, 2012.

[74] Gaëtan Benoit, Claire Lemaitre, Dominique Lavenier, Erwan Drezen, Thibault Dayris, Raluca Uricaru, and Guillaume Rizk. Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph. *BMC Bioinformatics*, 16(1):288, 2015.

[75] H Sarkar and R Patro. Quark enables semi-reference-based compression of RNA-seq data. *bioRxiv*, (July):1–7, 2016.

[76] J Pell, A Hintze, R Canino-Koning, A Howe, J M Tiedje, and C T Brown. Scaling metagenome sequence assembly with probabilistic de Bruijn graphs. *Proceedings of the National Academy of Sciences*, I(1):1–11, 2012.

[77] Rayan Chikhi and Guillaume Rizk. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. *Algorithms in Bioinformatics*, pages 236–248, 2012.

[78] Shaun D Jackman, Sarah Yeo, Lauren Coombe, and Rene L Warren. ABySS 2 . 0 : Resource-Efficient Assembly of Large Genomes using a Bloom Filter. *Genome Research*, pages 768–777, 2017.

[79] Rayan Chikhi, Antoine Limasset, Shaun Jackman, Jared T. Simpson, and Paul Medvedev. On the representation of de bruijn graphs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8394 LNBI, pages 35–55, 2014.

[80] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

[81] David Pellow, Darya Filippova, and Carl Kingsford. Improving Bloom Filter Performance on Sequence Data Using k-mer Bloom Filters. *Journal of computational biology : a journal of computational molecular cell biology*, 24(6):547–557, 2017.

[82] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. DeBGR: An efficient and near-exact representation of the weighted de Bruijn graph. *Bioinformatics*, 33(14):i133–i141, 2017.

[83] Ilia Minkin, Son Pham, and Paul Medvedev. TwoPaCo: An efficient algorithm to build the compacted de Bruijn graph from many complete genomes. *Bioinformatics*, 2016.

[84] Heng Li. Minimap and miniasm: Fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[85] Heng Li. Minimap2: fast pairwise alignment for long nucleotide sequences. 2017.

[86] Chen Sun, Robert S. Harris, Rayan Chikhi, and Paul Medvedev. AllSome Sequence Bloom Trees. *bioRxiv*, 2017.

[87] Brad Solomon and Carl Kingsford. Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees. *bioRxiv*, 2016.

[88] S. Halary, J. W. Leigh, B. Cheaib, P. Lopez, and E. Bapteste. Network analyses structure genetic diversity in independent genetic worlds. *Proceedings of the National Academy of Sciences*, 107(1):127–132, 2009.

[89] H. C. Neu. The Crisis in Antibiotic Resistance. *Science*, 257(5073):1064–1073, 1992.

[90] Tue Sparholt Jørgensen, Zhuofei Xu, Martin Asser Hansen, Søren Johannes Sørensen, and Lars Hestbjerg Hansen. Hundreds of circular novel plasmids and DNA elements identified in a rat cecum metamobilome. *PloS one*, 9(2):e87924, 2014.

[91] Aya Brown Kav, Goor Sasson, Elie Jami, Adi Doron-Faigenboim, Itai Benhar, and Itzhak Mizrahi. Insights into the bovine rumen plasmidome. *Proceedings of the National Academy of Sciences of the United States of America*, 109(14):5452–7, 2012.

[92] Sara El-Metwally, Magdi Zakaria, and Taher Hamza. LightAssembler: Fast and memory-efficient assembly algorithm for high-throughput sequencing reads. *Bioinformatics*, 32(21):3215–3223, 2016.

[93] S. Nurk, D. Meleshko, A. Korobeynikov, and P. Pevzner. metaSPAdes: a new versatile de novo metagenomics assembler. page arXiv:1604.03071, 2016.

[94] Dinghua Li, Chi Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak Wah Lam. MEGAHIT: An ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics*, 31(10):1674–1676, 2014.

[95] Rahul Nihalani and Srinivas Aluru. Effective Utilization of Paired Reads to Improve Length and Accuracy of Contigs in Genome Assembly. In *Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, pages 355–363, 2016.

[96] Andrey D. Prjibelski, Irina Vasilinetc, Anton Bankevich, Alexey Gurevich, Tatiana Krivosheeva, Sergey Nurk, Son Pham, Anton Korobeynikov, Alla Lapidus, and Pavel A. Pevzner. ExSPAnder: A universal repeat resolver for DNA fragment assembly. *Bioinformatics*, 30(12), 2014.

[97] Wenyu Shi, Peifeng Ji, and Fangqing Zhao. The combination of direct and paired link graphs can boost repetitive genome assembly. *Nucleic acids research*, page gkw1191, 2016.

[98] Isaac Turner, Kiran V. Garimella, Zamin Iqbal, and Gil McVean. Integrating long-range connectivity information into de Bruijn graphs. *bioRxiv*, 2017.

[99] Anthony M. Bolger, Alisandra K. Denton, Marie E. Bolger, and Bjoern Usadel. LOGAN: A framework for LOssless Graph-based ANalysis of high throughput sequence data. *bioRxiv*, 2017.

[100] David Paez-Espino, Emiley A. Eloe-Fadrosh, Georgios A. Pavlopoulos, Alex D. Thomas, Marcel Huntemann, Natalia Mikhailova, Edward Rubin, Natalia N. Ivanova, and Nikos C. Kyrpides. Uncovering Earth's virome. *Nature*, 536(7617):425–430, 2016.

[101] Laura A. Hug, Brett J. Baker, Karthik Anantharaman, Christopher T. Brown, Alexander J. Probst, Cindy J. Castelle, Cristina N. Butterfield, Alex W. Hernsdorf, Yuki Amano, Kotaro Ise, Yohey Suzuki, Natasha Dudek, David A. Relman, Kari M. Finstad, Ronald Amundson, Brian C. Thomas, and Jillian F. Banfield. A new view of the tree of life. *Nature Microbiology*, 1(5):16048, 2016.

[102] Evangelos Georganas, Aydn Buluç, Jarrod Chapman, Steven Hofmeyr, Chaitanya Aluru, Rob Egan, Leonid Oliker, Daniel Rokhsar, and Katherine Yelick. HipMer : An Extreme-Scale De Novo Genome Assembler. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis.*, pages 14:1–14:11, 2015.

[103] Jintao Meng, Bingqiang Wang, Yanjie Wei, Shengzhong Feng, and Pavan Balaji. SWAP-Assembler: scalable and efficient genome assembly towards thousands of cores. *BMC Bioinformatics*, 15(Suppl 9):S2, 2014.

[104] Anas Abu-Doleh and Umit V. Catalyurek. Spaler: Spark and GraphX based de novo genome assembler. *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, (C):1013–1018, 2015.

[105] Guy Bresler, Ma'ayan Bresler, and David Tse. Optimal assembly for high throughput shotgun sequencing. *BMC bioinformatics*, 14 Suppl 5(Suppl 5):S18, 2013.

[106] Govinda M Kamath, Ilan Shomorony, Fei Xia, Thomas A Courtade, and David N Tse. HINGE: long-read assembly achieves optimal repeat resolution. *Genome Research*, 27(5):747–756, 2017.

[107] Ilan Shomorony, Govinda M. Kamath, Fei Xia, Thomas A. Courtade, and David N. Tse. Partial DNA assembly: A rate-distortion perspective. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 1799–1803. IEEE, 2016.

[108] Sreeram Kannan, Joseph Hui, Kayvon Mazooji, Lior Pachter, and David Tse. Shannon: An Information-Optimal de Novo RNA-Seq Assembler. *bioRxiv*, 2016.

ליישומם השפעה רבה בתעשייה ובחקלאות .עם זאת ,אפיון שלהם באמצעות ריצוף מהדור החדש (NGS) נותר מאתגר, למרות הצניחה המהירה במחיר והעלייה בתפוקה עבור ריצוף. במחקר זה, אנו מנסים לשפר את יכולת אפיון פלסמידים על ידי פיתוח אלגוריתם חדש עבור הרכבה של אלמנטים מעגליים, ניצול גרפים של הרכבה המסופקים על ידי כלי להרכבה (de novo assembler) קונבנציונאלי ועימוד של קריאות קצוות-מזווגים, במטרה להרכיב רצפים מעגליים העשויים להיות פלסמידים, פאג'ים ואלמנטים מעגליים אחרים.

תוצאות :אנו מציגים את Recycler, הכלי הראשון שביכולתו לחלץ קונטיגים (contigs) מעגליים מנתונים של ריצופים עבור גנומים מיקרוביאליים מבודדים, פלסמידומים ומטאגנומים. אנו מראים כי Recycler מגדיל מאוד את מספר הפלסמידים האמיתיים ששוחזרו יחסית לגישות אחרות תוך שמירה על דיוק גבוה. אנו מראים מגמה זו באמצעות סימולציות של פלסמידומים, השוואות של חיזויים עם נתוני ייחוס עבור דגימות ממיקרובים מבודדים, והערכות של דיוק אפיון נתונים של מטאגנום. בנוסף ,אנו מספקים אימות על ידי הגברת דנ"א של 77 פלסמידים שנחזו על ידי Recycler מתוצאות ריצוף של סוגים שונים. בבדיקה זה הראה Recycler דיוק ממוצע של 89% על פני כל סוגי הנתונים - מבודדים, מיקרוביום, ופלסמידום.

3. **Faucet: streaming de novo assembly graph construction**

R. Rozov, R.Shamir, E. Halperin; Bioinformatics btx471, 2017

מוטיבציה :אנו מציגים את Faucet, אלגוריתם לבניית גרף הרכבה בשיטת זרימת נתונים (streaming algorithm) בשני מעברים. האלגוריתם בונה גרף הרכבה בהדרגה תוך מעבר על כל קריאה. אין צורך באחסון מקומי עבור כל הקריאות, והעיבוד עבור אותן קריאות מתבצע תוך כדי הורדת הנתונים ומחיקתן. אנו מדגימים פונקציונליות זו על-ידי בניית גרף הרכבה תוך כדי זרימת נתונים זמינים לציבור, ומבחינים כי היחס בין השימוש בדיסק לגודל נתונים גולמי פוחת ככל שרמת הכיסוי עולה.

תוצאות : Faucet משלב בין גרף de Bruijn שהושג מן הקריאה עם נתונים תיאוריים (metadata) נוספים הנגזרים מהקריאות. אנו מראים שנתונים תיאוריים אלו- אשר מורכבים מרמות הכיסוי שנספרו בצמתי הגרף, וחיבורים המגשרים בין זוגות צמתים - מכילים את רב המידע הנחוץ לשם הרכבה. לשם כך, אנו מדגימים יכולת ניקוי גרפי הרכבה של מטאגנומים, עם שיפור משמעותי בהמשכיות הרצפים שנוצרו תוך שמירה על דיוקם. השווינו את השימוש במשאבים ובאיכות הרכבה של Faucet לכלי הרכבה מובילים של מטאגנומים ושל גנומים. השימוש ב- Faucet מצמצם את זמן ההרכבה ומקטין את שטח האחסון בכסדר גודל לעומת כלי הרכבת מטאגנומים יעודיים כגון MetaSPAdes ו- Megahit, תוך שיפור השימוש בזיכרון שלהם; ישנה התאמה כללית ברמת הביצועים של Faucet לשיטות הרכבה אשר פועלות תוך מיקסום יעילות המשאבים - כגון - Minia, ו-LightAssembler. עם זאת, על מטגנומים שנבדקו, התפוקות בשיטת Faucet היו כ- 110%-14 גבוהות יותר לפי קריטריון NGA50 ממוצע לעומת Minia ופי 2-11 גבוהות לפי אותו קריטריון לעומת LightAssembler, כלי ההרכבה היחיד הזמין בשיטת זרימת נתונים.

תקציר המאמרים הכלולים בתזה

להלן תקצירי המאמרים עליהם מבוססת עבודה זו :

1. **Fast lossless compression via cascading Bloom filters.**

R. Rozov, R.Shamir, E. Halperin; BMC Bioinformatics 2014, 15 (Suppl 9):S7.

רקע :נתונים מניסויים גדולים בשיטות ריצוף מהדור החדש (NGS) יוצרים אתגרים הן מבחינת העלויות הכרוכות באחסון והן בזמן הנדרש בהעברת הקובץ .לפעמים ניתן לאחסן רק סיכום רלוונטי ליישומים מסוימים, אך בדרך כלל רצוי לשמור את כל המידע הדרוש בכדי לבחון את תוצאות הניסוי בעתיד .לכן, הצורך בשיטות יעילות לדחיסה ללא אובדן מידע (lossless) עבור קריאות NGS עולה. הוכח כי שיטות דחיסה ספציפיות ל NGS -יכולות לשפר את התוצאות על שיטות דחיסה גנריות, כגון אלגוריתם למפל-זיו, טרנספורמצית בורוס-וילר, או קידוד אריתמטי. כאשר גנום ההשוואה זמין, ניתן לבצע דחיסה יעילה, תחילה על ידי עימוד הרצפים לרצף ההשוואה, ולאחר מכן קידוד כל רצף באמצעות מיקום העימוד בשילוב עם ההבדלים בקריאות יחסית לרצף ההשוואה. שיטות מבוססות ייחוס לרצף השוואה הוכחו כבעלות יכולת דחיסה טובה יותר מאשר השיטות שלא מבוססות ייחוס לרצף השוואה, אך פעולת העימוד שהן דורשות לוקחת כמה שעות של חישוב עבור אוסף נתונים טיפוסי, בעוד ששיטות ללא ביצוע ייחוס לרצף השוואה, מאפשרות בד״כ דחיסה תוך דקות.

תוצאות :אנו מציגים גישה חדשה אשר משיגה דחיסה יעילה מאוד באמצעות ייחוס לרצף השוואה, אבל עוקפת לחלוטין את הצורך לעימוד, תוך הפחתה גדולה בזמן הדרוש לביצוע הדחיסה. בניגוד לשיטות מבוססות ייחוס לרצף השוואה, אשר תחילה מבצעות עימוד של קריאות לגנום, אנו מבצעים גיבוב (hash) עבור כל הקריאות לתוך מסנן בלום (Bloom) filter לשם קידוד ומבצעים שאילתות על אותו מסנן בלום לשם פענוח. דחיסה נוספת מושגת באמצעות מפל של מסננים אלו.

מסקנות :השיטה שלנו, הנקראת BARCODE, מהירה יותר בכסדר גודל משיטות המבוססות על ייחוס לרצף השוואה, תוך דחיסה של כסדר גודל יותר משיטות ללא ייחוס לרצף השוואה, על פני טווח רחב של רמות כיסוי הרצפים. בכיסוי גבוה (פי 100-50), בהשוואה לשיטות הדחיסה הטובות ביותר BARCODE, חוסך 90%-80 של זמן ריצה תוך הגדלת שטח מועטה.

2. **Recycler: an algorithm for detecting plasmids from de novo assembly graphs**

R. Rozov, A.B. Kav, D. Bogumil, N. Shterzer, E. Halperin, I. Mizrahi, R. Shamir; Bioinformatics 2017; 33 (4): 475-482.

מוטיבציה :פלסמידים ואלמנטים ניידים אחרים של DNA מהווים תרומה מרכזית לאבולוציה חיידקית וחדשנות גנומית. לאחרונה, נמצא כי לאלמנטים אלו תפקידים חשובים הן ביצירת עמידויות לאנטיביוטיקה והן ביצירת מטבוליטים אשר

באמצעות נתונים מדומים ואמתיים. יתר על כך, מתוך פלסמידים שנובאו ונבדקו באמצעות Recycler, כמעט 90% אומתו בבדיקות מעבדה.

לבסוף, בפרק 4, אנו מציגים את Faucet, אלגוריתם להרכבה המבצע בניית גרף באמצעות זרימת נתונים. אלגוריתם זה מאפשר למידע חיוני להיקלט ולעבור קידוד ביעילות תוך זרימת הנתונים. הראינו כי שיטה זו דוחסת ביעילות נתוני קלט, וכי דחיסה זו הופכת יעילה יותר ככל שרמת כיסוי הקריאות עולה. הגרף ש Faucet מקודד אליו מאפשר תשאול לשם הרכבה, ומציג אמצעי דחיסה חדש שייתכן שיהיו לו יישומים נוספים. בהשוואה לשיטות הקיימות, Faucet יצר הרכבות רציפות יותר מאשר שיטות חסכניות מבחינת משאבים, כגון minia [77] ו-LightAssembler [92] ביעילות ומהירות הגבוהות מהשיטות הטובות ביותר כיום להרכבת מטגנומים, כגון metaSPAdes [93] ו [94] MegaHit . מבחינת שימוש בדיסק, ביצועי FAUCET היו דומים לאלה של LightAssembler , כלי ההרכבה היחיד הנוסף העובד בשיטת זרימת נתונים [92], אך עם זאת Faucet יצר תוצאות הרבה יותר רציפות.

בשיטת הריצוף מבוסס סינתזה (SBS) [7]. בנוסף, מקטעי ה-DNA עוברים ריצוף בנפרד בטכנולוגיות אלו, ללא צורך בהגברה בעזרת PCR, ולכן יש מקור אחד פחות להטיות תלויות רצף. נכון להיום, החסרונות העיקריים בשיטות אלו לעומת ה-SBS הינם בעלות מחיר גבוהה לבסיס ושיעור שגיאות הגבוה בהרבה בקריאות, אך עם זאת הן אוחזות ביתרונות עצומים עבור הרכבה ללא ידע מוקדם (de novo assembly) [8], בהבדלה בין תעתיקים קרובים הקשורים זה לזה, הנוצרים בריצוף mRNA (RNA-Seq) [9,10], או בהבחנה בין זנים קרובים של וירוסים בריצוף מטגנומים [11,12].

השינוי המהיר בנוף הטכנולוגי יצר אתגרים ביואינפורמטיים חדשים. עליית מספר הרצפים לדגימה יחד עם העלייה במספר וגודל המחקרים בעקבות עלות ריצוף זולה יצרו הצפה של נתונים. בכדי לשכלל את יכולת משימות הניתוח, התפתחו יחדיו אלגוריתמים וטכנולוגיות ריצוף חדשים. גם לתחומים אשר נחשבו בשלים כגון עימוד רצפים, בו אלגוריתמים אופטימליים ידועים עבור עימוד רצפים בודדים [13,14], וקיימים אלגוריתמים היוריסטיים יעילים בעימוד רצפים בודדים למאגרי רצפים גדולים [15], נדרשו חידושים על מנת להתמודד עם האתגרים החדשים, כגון עימוד מיליוני רצפים קצרים מאוד לגנומים של יונקים [16,17], חיפוש קבצי נתונים רלוונטיים בקרב האלפים אשר זמינים לציבור (כאשר כל אחד מהקבצים מורכב ממיליוני רצפים) [18,19,20], וקירוב יעיל של עימוד כל הזוגות האפשריים בין מיליוני קריאות במטרה לבצע הרכבה [21,22]. כמו כן, ריצוף זול איפשר ביצוע הרכבה עבור מספר רב יותר של גנומים, גנומים ארוכים יותר, ואוספים של מינים או תעתיקים דרך ריצוף מטגנומי ו- RNA-Seq בהתאמה. האתגרים בתחום זה דרשו שטח אחסון גדול יותר כדי לאפשר ייצוג של אוספים גדולים של רצפים בו זמנית. בשני המקרים, הן בהרכבת והן בעימוד רצפים, טכניקות חדשות למפתוח או לסידור הרצפים מחדש הוכחו כהכרחיות.

טכניקות אלו הינן במוקד תזה זו. אנו מתחילים בתיאור מפורט של טכנולוגיות ריצוף וקצב יצירת הנתונים שלהם. לאחר מכן, אנו מאפיינים את האתגר שנתונים אלה מטילים על אלגוריתמים מסורתיים של מחרוזות. לבסוף, אנו דנים בגישות מודרניות לבניית מפתח מחרוזות (string indexing), לפני תיאור תרומתנו אשר מבוססות עליהן. לבסוף אנו מציגים פרספקטיבה לגבי ההתפתחויות העתידיות של טכנולוגיות ריצוף והכלים אשר יהיו נחוצים לצורך ניתוח התפוקה שלהן.

תוצאות

בפרק 2, אנו מציגים את BARCODE, אלגוריתם דחיסה מהיר של קריאות קצרות המבוסס על מסנן בלום (Bloom filter) המבצע קידוד של קריאות ביחס לרצף השוואה ידוע. BARCODE עוקף את הצורך בעימוד הקריאות על ידי גיבוב (hashing) הקריאות לתוך מסנן בלום ומונע את המגבלות אשר טמונות בפילטר בלום בעזרת מעקב מדוקדק אחר קריאות חוזרות ודחיסתן, בנוסף למעקב ודחיסת קריאות שלא תואמות לגמרי את רצף ההשוואה. האלגוריתם הודגם כדוחס מהיר יותר משיטות המבוססות על ייחוס לרצף השוואה, וכמו כן בעל יחסי דחיסה טובים יותר מאשר שיטות אשר אינן מבוססות על ייחוס לרצף השוואה.

בפרק 3, אנו מציגים את Recycler, אלגוריתם להרכבת רצפי פלסמיד. Recycler מזהה מעגלים השייכים לפלסמידים בגרף ההרכבה שנוצר על ידי כלים זמינים להרכבה. בעזרת פעולה זו, ביכולתו לשחזר פלסמידים רבים יותר מאשר שיטות זמינות בעבר. Recycler הראה יכולות רגישות ודיוק גבוהות יותר מאשר שיטות מתחרות, בהתבסס על הערכות

תמצית

נפח הנתונים שנוצר בשיטת הריצוף העמוק גדל באופן מעריכי בעשור באחרון. בשיטה זו, אשר מחליפה את שיטת סנגר האיטית, הריצוף נעשה בצורה מקבילה של מיליוני מקטעים קצרים, המרוצפים בו זמנית. כתוצאה משינוי זה תפוקת הריצוף התעצמה עד כדי כך שהמחיר לריצוף בסיס (נוקלאוטיד) צנח בחמישה סדרי גודל. הזמן הנחוץ בכדי לבצע תהליכי עיבוד כגון הרכבת הגנום ועימוד רצפים יגדל בצורה מעריכית גם כן, במידה ואותם אלגוריתמים שנועדו לעיבוד רצפים קצרים (או קטני נפח) יופעלו על נתוני הריצוף העמוק. בעבודת תזה זו אנו מציגים שיטות חדשות, אשר נועדו להתמודד עם האתגרים שמציב עידן הריצוף העמוק. אנו משתמשים במבני נתונים מתקדמים כדי לבצע דחיסה ומפתוח של נתוני הרצפים, במטרה לאפשר תשאול יעיל תוך הורדת נפח אחסון. בהקשר לדחיסת הרצפים, אנו מדגימים גישה מהירה יותר מהשיטות הקיימות כיום לדחיסת מקטעי רצפים ביחס לרצף ההתייחסות (reference), על ידי הצגת הרצפים תוך שימוש בשיטת מסנן בלום (Bloom filter). אנו מציגים שני שיפורים בהקשר להרכבת הגנום: פיתחנו שיטה חדשה להרכבת רצפים מעגליים, תוך שימוש בגרפים של ההרכבה כמפתח. כמו כן, פיתחנו אלגוריתם זרימה (streaming), לבניית גרף ההרכבה, במטרה להוריד את זמן הבנייה, הזיכרון והשימוש בדיסק. אנו מראים שמבנה זה מאפשר שמירה של עיקר תכונות הרצפים, דבר המאפשר הרכבת גנום רציפה ואיכותית.

רקע כללי

יכולתו של המין האנושי לקרוא דנ"א השתפרה באופן דרמטי בעשור הראשון של המאה העשרים ואחת. העלויות העצומות שנדרשו עבור השלמת הריצוף והרכבת הרצף של הגנום האנושי הראשון באמצעות שיטת סנגר, זירזו את פיתוחן של טכנולוגיות שונות במטרה להאיץ את מהירות הריצוף, להוזיל עלויות, ולהגדיל את התפוקה. מספר חברות הציגו טכנולוגיות מתחרות בכדי לענות על צרכים אלה. המשותף לחברות אלו, הוא שימוש בטכנולוגיות "ריצוף עמוק" או "ריצוף הדור הבא". אחת מהחברות הללו, Illumina, שולטת כעת ב-70% מהשוק [1]. הדבר התאפשר בזכות טכנולוגית ריצוף על ידי סינתזה, (SBS), שפיתחה חברה קטנה יותר בשם Solexa בשנת 2006 [2].

הפיתוח של ריצוף עמוק במהלך העשור האחרון הוביל לירידה במחיר ביותר מחמישה סדרי גודל, ועלייה בתפוקה בשבעה סדרי גודל ביחס לריצוף בשיטת סנגר [3] [Figure 1.1]. כיום, עולה בערך 1000 דולר לרצף גנום אנושי, לעומת 200 מיליון דולר שריצוף כזה עלה בשנת 2001. לירידה זו במחיר היו השפעות עמוקות על המחקר הביולוגי. לדוגמה, כעת ניתן לבצע מחקרים על מאות אלפי אנשים כדי לאפיין היטב שונות בין בני אדם [4]. כמו כן, פותחו כמעט מאה טכניקות ניסוי חדשות המבוססות על ריצוף [5]. במקרים מסוימים, הריצוף תפס את מקומה של הטכנולוגיה הקיימת עקב השגת רזולוציה גבוהה הרבה יותר, כמו במקרה של ריצוף mRNA למדידת שעתוק (נקרא RNA-Seq) אשר החליף את שיטת שבבי הדנ״א (microarrays). במקרים אחרים, ריצוף עמוק אפשר סוגים של ניסויים שלא התאפשרו בעבר בתפוקה גבוהה, כגון Chromosome Conformation Capture (C3), אשר פותח לצורך חקירת מבנה הגנום בתלת מימד [6].

עם הזמן, פותחו יישומים חדשים עבור שיטת ריצוף עמוק, לצד טכנולוגיות חדשות אשר צמחו במטרה להתמודד עם חסרונותיה. טכנולוגיות ריצוף של הדור השלישי יוצרות קריאות אשר אורכן גדול פי עשרה עד אלף יותר מהקריאות

אוניברסיטת תל-אביב TEL AVIV UNIVERSITY

הפקולטה למדעים מדויקים ע״ש ריימונד ובברלי סאקלר
בית הספר למדעי המחשב ע״ש בלבטניק

על הפחתת הסיבוכיות של ניתוח נתוני ריצוף עמוק

חיבור לשם קבלת תואר ״דוקטור לפילוסופיה״

מאת **רועי רוזוב**

בהנחייתו של פרופ׳ **רון שמיר**
ופרופ׳ **ערן הלפרין**

הוגש לסנאט של אוניברסיטת ת״א

נובמבר 2017