Sackler Faculty of Exact Sciences,
Blavatnik School of Computer Science

Methods for efficient analysis of large sequencing experiments

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

David Pellow

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

Submitted to the Senate of Tel-Aviv University
September 2022

Acknowledgements

This thesis represents the bulk of my research from a long and significant part of my life. The work was often challenging, and I would not have gotten here without help from many people.

First and foremost I would like to thank my advisor Professor Ron Shamir. I am very grateful to have had the opportunity to learn from and work with him. Without his guidance, support and advice, none of this would have been possible.

I am also indebted to the many current and former members of the Shamir Lab for help, advice, and fruitful discussions along the way. Especially to Roye Rozov, Nimrod Rappoport, Lianrong Pu, and Hagai Levy for their helpful insights. I would also like to thank Gilit Zohar-Oren for always helping with anything we need and ensuring the process went smoothly.

In the course of my PhD I was privileged to take part in excellent collaborations, which resulted in the research presented in this thesis. I am grateful for all of my collaborators: Carl Kingsford and Guillaume Marçais from the Kingsford Lab, Alvah Zorea and Itzik Mizrahi from the Mizrahi Lab, Yaron Orenstein and Lior Kotlar from the Orenstein Lab, Abhinav Dutta, and Dan Flomin.

The research presented in this thesis was funded in part by the Israel Ministry of Immigrant Absorption, by the Edmond J. Safra Center for Bioinformatics at Tel Aviv University, the Israel Science Foundation, and Len Blavatnik and the Blavatnik family fund.

Lastly, I would like to thank my family: My parents, for their constant support; my wife Emunah, for everything over the last five years; my kids, without whom it would have been a shorter and less fun period.

Preface

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

1. Orenstein, Y.*, Pellow, D.*, Marçais, G., Shamir, R., & Kingsford, C. (2017). Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLoS Computational Biology*, *13*(10), e1005777.

2. Pellow, D., Mizrahi, I., & Shamir, R. (2020). PlasClass improves plasmid sequence classification. *PLoS Computational Biology*, *16*(4), e1007781.

3. Pellow, D., Zorea, A., Probst, M., Furman, O., Segal, A., Mizrahi, I., & Shamir, R. (2021). SCAPP: an algorithm for improved plasmid assembly in metagenomes. *Microbiome*, *9*(1), 1-12.

In addition, contributions were made to two articles, and another preprint is under revision. The articles are:

- Marçais, G., Pellow, D., Bork, D., Orenstein, Y., Shamir, R., Kingsford, C. (2017). Improving the performance of minimizers and winnowing schemes. *Bioinformatics, 33*(14), i110-i117.

- Flomin, D., Pellow, D., Shamir, R. (2022). Data set-adaptive minimizer order reduces memory usage in k-mer counting. *Journal of Computational Biology 29*(8), 825-838.

The preprint is:

Dutta, A.*, Pellow, D.*, Shamir, R. (2022). Parameterized syncmer schemes improve long-read mapping. *bioRxiv*.

Abstract

The volume of high-throughput sequencing data is constantly growing, and new sequencing technologies such as long-read sequencing and new data types such as metagenomic sequencing are frequently introduced. As a result, constant computational innovation is needed to improve or modify basic processing steps such as read assembly and alignment in order to handle new types of data while also improving runtime and memory efficiency. In this thesis I present our work contributing to this challenge. First, we introduced the notion of the Universal Hitting Set (UHS), a compact set of k-mers that can be used to efficiently represent longer sequences. This set can be used to define a low density seed selection scheme that could improve the memory efficiency of a large range of basic sequence processing tasks. We developed a new heuristic for constructing compact UHS, and showed its advantage over extant approaches. Second, we developed methods to enable the assembly of complete plasmid sequences from metagenomic sequencing experiments. This required us to develop a better plasmid sequence classifier, and to incorporate background biological knowledge in an efficient manner to limit false positive assemblies. The resulting tools enable microbiologists to study the genetic content, population structure and ecology in plasmids across metagenomic samples.

Contents

Chapter 1

Introduction

The analysis of high-throughput sequencing data has enabled a rapid increase in biological discovery [52, 118, 14, 76]. Due to large and increasing data volumes from high-throughput sequencing, computational analyses have become more challenging and must be more memory and runtime efficient. The challenges are compounded by ever-advancing technologies and the new analyses that they enable. For example, third-generation long-reads [65], high throughput single-cell sequencing [37], hi-C [10], and optical mapping [132] are new high throughput sequencing methods and technologies. These high throughput technologies allow scientists to interrogate not just the genome sequence and RNA expression but also the 3D structure of the genome [13], the community structure of diverse and complex microbiomes [138], epigenetic interactions and regulation [40, 44], cell sub-types [125, 122], trajectories of gene expression programs in cells throughout their lifespan [16] and endless other biological questions.

The ability to analyse ever-growing sequencing data has depended on theoretical algorithmic advances going back decades. Some notable advances on this front were the introduction of the suffix array [70] and Burrows Wheeler Transform [57] for indexing and alignment, BLAST [1] for local alignment, Bloom filters [12] for approximate k-mer set containment queries [117], the de Bruijn graph for assembly [94], minimizer sequence sketches [102], and MinHash for approximate sequence matching [87], among many others. The latest generation of sequence analysis tools have built on even more complex algorithmic work.

In addition to the introduction of elegant theoretical algorithms and data struc-

tures, advances in high-throughput analysis have depended on good software engineering, effective heuristics, and implementation tricks and details that squeeze ever more computational efficiency out of these methods. Molecular biology, with more exceptions than rules [53], complicates matters further. Effective tools may have to leverage prior biological knowledge and models, messy or ad hoc heuristics, and special cases on top of any algorithmic advances.

In this thesis I present my work on both fronts – basic theoretical advances that improve the computational efficiency and performance of sequence analysis tools, and efficient implementations of bioinformatics tools for plasmid reconstruction that leverage biological knowledge to increase effectiveness. In the first case, we worked on the universal hitting set (UHS) problem, developed an algorithm, DOCKS, that solves it heuristically, and used UHS produced by DOCKS to create k-mer selection schemes with lower density than other minimizer schemes. In the second case, I worked on plasmid sequence classification and reconstruction in metagenomic samples. I introduced the PlasClass classifier and the SCAPP plasmid assembler, which leverages prior knowledge of plasmid biology to be more effective.

In the following sections in the introduction I provide the necessary background and definitions on the topics of efficient k-mer selection, classification of metagenomic sequences, and plasmid reconstruction. The subsequent chapters of the thesis present my papers listed in the last section of the introduction. The final chapter of the thesis offers a discussion of the papers, their impact, and possible future directions.

1.1 k-mer selection schemes

1.1.1 Background

A *k-mer* in a sequence S over an alphabet Σ is a consecutive substring of S of length k (in our case the nucleotide alphabet A, C, G, T). S can be represented by all of its overlapping k-mers: $S_1...S_k$, $S_2...S_{k+1}$, ..., $S_{|S|-k+1}...S_{|S|}$.

A *k-mer selection scheme* is a function that selects a subset of the positions in a sequence: $f_k(S) : \Sigma^* \rightarrow \mathcal{P}(\{1, ..., |S| - k + 1\})$, where $\mathcal{P}$ represents the powerset. In most cases the identity of the k-mer at a selected index is also of interest, and

the scheme can be defined: $f_k(S) : \Sigma^* \to \{(i, s)\}$, $i \in \{1, ..., |S| - k + 1\}, s \in \Sigma^k$. In some cases, when the selected k-mers are used as a hash of S, only the identity of the selected k-mers matters: $f_k(S) : \Sigma^* \to \Sigma^k$. In practice, for the DNA alphabet it is often desirable to treat a sequence and its reverse complement as a single entity, thus *canonical* k-mers, that represent each k-mer by the minimum of itself and its reverse complement are considered. For k-mer x with reverse complement $\bar{x}$ and order over the k-mers o, $Canonical(x) = min_o\{x, \bar{x}\}$, i.e. the minimum of the two under a k-mer order o.

I will explain key properties of selection schemes using two simple examples. A very simple selection scheme is $f_k(S) = \{(i, S_i..S_{i+k-1}) | i \mod m = 0\}$, i.e. to select every m-th position. The scheme has a *window guarantee*, meaning that any sequence of w consecutive k-mers, for some fixed w (in this case $w = m$), is guaranteed to contain a selected index. Note that the final $k-1$ positions cannot be selected as they do not start a full k-mer. This scheme has *density* $= 1/m$, meaning that on average one index is selected from every m positions. This is the minimal density possible for a scheme with a window guarantee of length m. However, this scheme is lacking the key property of *consistency*: two nearly identical sequences may have completely different k-mers selected. For example $S_2...S_{|S|}$ will have a non-overlapping set of k-mers selected from it even though it is a substring of S, sharing identical sequence except for the first position of S. In almost all applications consistency is a necessary property of a selection scheme.

Another simple scheme is: $f_k(S) = \{(i, S_i..S_{i+k-1}) | h(S_i..S_{i+k-1}) < H/\delta\}$ for a given $0 < \delta \leq 1$, and hash function $h : \Sigma^k \to [0, H]$, δ is called the *compression rate*. Note that the compression rate is the inverse of density; in expectation over an infinitely long random sequence $1/\delta$ of the positions will be selected. However, this scheme does not have a window guarantee and arbitrarily long sequences may contain no selected k-mer. This scheme is *local* (referred to as "1-local" in [112]), meaning that only the identity of the k-mer itself determines whether it is selected. Non-local schemes select k-mers as a function of their sequence context. In the first example scheme above, the positions selected depend on the entire sequence, as would be the case a scheme that selects k-mers based on their frequency in the sequence, for example. A w-local scheme selects k-mers as a function of a sequence context of w k-mers. For example, a scheme that yields a window guarantee for length w may select a k-mer from a window of w k-mers depending on the contents of that window. As a 1-local scheme, the current example has better *conservation*

than a scheme that selects k-mers based on a longer window in which they appear. This means that selected k-mers are more likely to be conserved under mutation of the sequence (or sequencing errors). Specifically, a mutation anywhere in the w-long window may change the k-mer selected by a w-local scheme, while in the 1-local scheme, the selected k-mer will be conserved unless the mutation appears in it, which is less likely.

Selection schemes have many practical applications. They can be used to select seeds from a sequence for indexing [129, 128, 75] and alignment [91, 112, 108, 48, 61], as a hash of their underlying sequence (e.g. for sequence binning [22, 32, 35, 28, 63, 86, 75] or locality sensitive hashing [47]), to sparsify the sequence [30, 128, 99], or as a sequence sketch [47]. (Sketching is a broader topic. For a practical review of sequence sketching see [106] and for a review of sketching and other related methods see [74]) In the following subsections I introduce some practical selection schemes and their applications.

1.1.2 Minimizer schemes

The most commonly used sequence selection schemes in computational biology applications are minimizer schemes. Minimizers were first defined by Schleimer et al. [109], for the purpose of plagiarism detection (they refer to their scheme as the winnowing algorithm). They were introduced into bioinformatics by Roberts et al. [102] as a method to find similar sequences. A minimizer scheme selects the minimum k-mer from every window of w contiguous k-mers in a sequence according to some order, o, over the k-mers. An example minimizer scheme is shown in Figure 1.1.

Minimizers have a window guarantee for windows of length w by construction and are w-local. Because the minimum k-mer is likely to be maintained across multiple overlapping windows, minimizer schemes have relatively low density. The expected density was approximated to be $2/(w+1)$ [102], but the assumptions of this approximation do not always hold and lexicographic orders achieve slightly higher expected density [73]. In practice, nucleotides are not randomly distributed in real genetic sequences, and minimizer schemes using a random ordering of the k-mers have much better density on actual sequences than the lexicographic order [73]. Some works have attempted to improve the performance of the lexicographic order by re-ordering certain k-mers [102, 28]. In Marçais et al. [73] we built on the selection

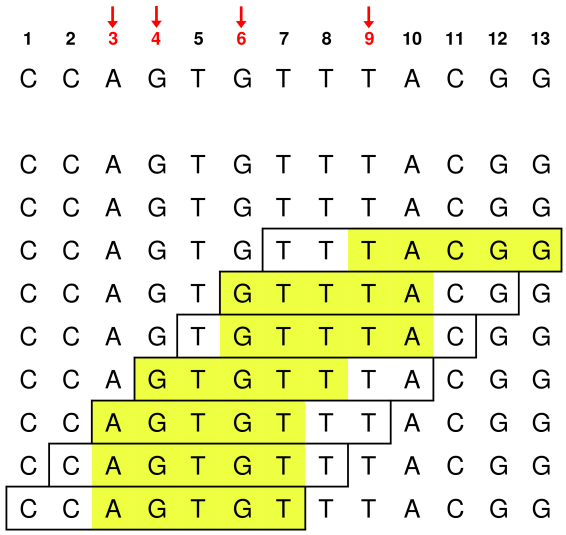


Figure 1.1: **Minimizer schemes.** An example minimizer scheme with $w = 3$, $k = 5$, and lexicographic ordering. All overlapping windows of 3 5-mers are shown. The minimizer is the least (lexicographically smallest) 5-mer in every window, highlighted in yellow. Selected positions are marked in red in the underlying sequence at the top. Note that for simplicity, only k-mers on the forward strand are considered in this example. Source: [91].

scheme introduced in the first paper included in this thesis [88] to define a minimizer scheme with expected density $2(1 - SP(\mathcal{U}))/(w + 1)$, where $\mathcal{U}$ is the scheme and $SP(\mathcal{U})$ denotes the *sparsity*, or fraction of all possible $w + 1$ long windows that contain more than one k-mer selected by $\mathcal{U}$. In practice this minimizer scheme also has lower density on real genomic sequences than minimizers with random ordering. Other works have used the k-mer frequency in a specific sequence or sample to define the minimizer order [49, 22] although the density of these minimizer schemes was not explored.

1.1.3 Universal hitting sets

The universal hitting set problem, introduced in the first paper of this thesis [88], seeks a minimal sized subset of k-mers $\mathcal{U}_{k,L} \subseteq \Sigma^k$ such that every sequence of length L contains at least one k-mer from $\mathcal{U}_{k,L}$. Any set $\mathcal{U}_{k,L}$ of k-mers hitting all L-long sequence is known as a *universal hitting set*, abbreviated UHS. There is no known efficient general construction for a UHS of optimal size, and in [88] we proposed a

heuristic algorithm, named DOCKS, to construct a small UHS for given k and L.

A selection scheme that selects k-mers from a UHS $\mathcal{U}_{k,L}$ will have a window guarantee for windows of length $w = L-k+1$, and will have low expected density if the UHS is small. Simply selecting every k-mer from $\mathcal{U}_{k,L}$ yields a 1-local scheme. A lower density could be achieved by assigning an order over k-mers in $\mathcal{U}_{k,L}$ and only picking the minimum element from $\mathcal{U}_{k,L}$ in every window of length $w = L - k + 1$. The expected density and improvement in actual density on real genomic sequences from this minimizer order were shown in [73] and discussed above.

1.1.4 Other selection schemes

Other selection schemes have been introduced in addition to those mentioned thus far. In this section I will briefly describe some of them, with more detail on methods I contributed to that are not included in the papers in this thesis.

Syncmers [29] are a 1-local scheme that selects k-mers that have their s-long minimizer (for $s < k$) at a specific position. *Open syncmers* have a single position at which the s-minimizer may appear and *closed syncmers* select a k-mer if its s-minimizer is in the first or last position. An example syncmer scheme is shown in Figure 1.2. As a 1-local scheme, syncmers have better conservation than minimizers and thus may be better suited to tasks such as sequence-to-sequence alignment between species or strains under mutation, or error-prone long-read mapping. Closed syncmers have a window guarantee with $w = k - s$, as the s-minimizer of a sequence of $k - s$ k-mers must appear as the first or last k-mer of one of them. Shaw and Yu [112] developed a theoretical framework to analyse the conservation of syncmers and implemented open syncmers in the minimap2 [61] read mapper. We generalized the notion of syncmers to k-mers with s-minimizer occurring in one of a set of arbitrary positions, named a *parameterized syncmer scheme* (PSS) [91]. We extended the theoretical analysis to the distribution of conserved k-mers selected by a PSS to select the best multiparameter scheme and implemented PSS in the latest version of minimap2 [62] and Winnowmap2 [48].

Improving minimizer schemes by modifying k-mer order has been the focus of some works aimed at achieving balanced bins when sequences are hashed to bins based on their minimizers. This is an important step in many k-mer counters that count each bin individually in order to reduce memory usage. For example

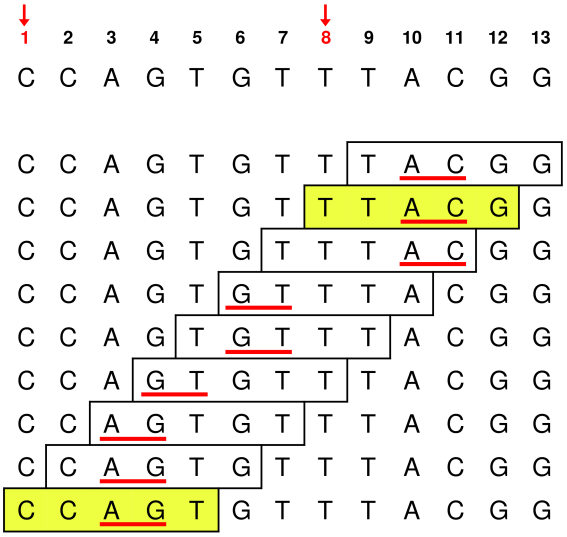


Figure 1.2: **Syncmer schemes.** An example syncmer scheme with $k = 5$, $s = 2$ and parameter (also called offset) $x_1 = 3$ using lexicographic ordering. The 2-minimizer of every 5-mer is underlined in red. k-mers with their s-minimizer at position x_1 are syncmers and highlighted in yellow. Selected positions are marked in red in the underlying sequence at the top. Note that only k-mers on the forward strand are considered in this example. Source: [91].

KMC2 [28] uses a re-ordered lexicographic order that moves k-mers that frequently appear in real genomes later in the order. Nyström-Persson et al. combined a UHS and frequency based order [86]. In Flomin et al. [35] we directly optimized the order to result in lower maximum bin size in an iterative, heuristic manner. Each k-mer was assigned an initial value corresponding to its position in the re-ordered lexicographic order of KMC2. Then, in each iteration, the bin size assigned to each k-mer was estimated by a sample from the sequences being binned. The k-mer with the largest bin is re-ordered by increasing its value by an additive penalty.

Another type of selection scheme is *sequence specific*, i.e., the selection scheme is tailored to a specific sequence and designed to have desirable properties on it. For example, AdaOrder [35] optimizes the order to achieve balanced bins for a specific order, and the frequency order [22, 49] uses a k-mer's frequency in a set of sequences to assign its rank in the order. Another sequence specific scheme is polar set based minimizers [137], which define a minimizer scheme using a set of k-mers with density minimized on a specific sequence. DeepMinimizer [42] uses deep learning to minimize the divergence between a low density template and a consistent scoring scheme to

construct a sequence specific low density scheme efficiently, even for large k.

Other selection schemes may not use k-mers and instead select subsequences from a sequence that are not contiguous or not of fixed length. For example, *strobemers* link between two selected syncmers that are some minimum and maximum distance from each other w_{min}, w_{max}, thus selecting gapped subsequences of length $2k$ [108]. Gapped minimizers are also used in Kraken2 [128] where k-mers are hashed to a binary string and every other bit from the 2nd least significant is masked up to s masked bits.

1.1.5 Applications of selection schemes

Many applications of k-mer selection schemes have been mentioned above. Here I list applications of a variety of selection schemes to highlight the importance of selection schemes across many areas of bioinformatics.

Lexicographic minimizers were first introduced to reduce memory usage for sequence similarity search [102]. Disk-based k-mer counters divide k-mers into bins based on their minimizer. Examples include MSPKmerCounter, which uses lexicographic minimizers [63], and KMC2 [28] and Gerbil [32], which use re-ordered lexicographic minimizers. minimap [60], minimap2 [61] and miniasm [60] use random minimizers as alignment seeds for long read mapping and assembly. Random orders are used in most other applications as well: MashMap [47] uses MinHash Jaccard approximation computed over random minimizers to efficiently find approximate long read mappings to a large database; kraken [129] uses random minimizers to store k-mers with the same minimizer in contiguous chunks of memory for metagenomic read classification; kraken2 [128] uses gapped, random minimizers as the sequence similarity seeds for the same task; ntJoin [25] creates graphs representing adjacent random minimizers for lightweight assembly scaffolding; MBG [99] represents a de Bruijn graph in minimizer space for lightweight assembly of hiFi reads; BLight [75] uses random minimizers to hash k-mers in an exact associative k-mer index. Frequency based minimizers were used to bin k-mers for counting as a preprocessing step to assembly in bcalm2 [22], and Winnowmap [49] re-weighted the most frequent k-mers to select minimizers as seeds for long read mapping. The Discount k-mer counter [86] used a combination of a UHS-based and frequency-based orders for minimizers to achieve evenly sized k-mer bins for counting. AdaOrder [35] directly

optimized the minimizer order to achieve evenly sized k-mer bins for counting.

Syncmers and PSS are used in long-read mapping and were shown to select alignment seeds that are better conserved than minimizers [112, 91]. Strobealign [108] uses linked syncmers for fast read alignment. Schemes selecting k-mers from a fixed subset of the universe of k-mers have been given different names: mincode submers [29], fractional minhash [46], universe minimizers [30], hash-based subsampling [128], and downsampling [91]. These schemes have been used on their own or in combination with other schemes such as minimizers or syncmers in a variety of applications including lowering memory usage for metagenomic read classification in Kraken2 [128]; reducing density of alignment seeds in long read mapping [91]; efficiently estimating taxonomic composition of metagenomes [46]; and very lightweight assembly of HiFi reads and pangenomes [30].

1.1.6 My contribution to k-mer selection schemes

We propose an alternative paradigm that can lead to substantial further improvement in these and other tasks. For integers k and $L > k$, we say that a set of k-mers is a universal hitting set (UHS) if every possible L-long sequence must contain a k-mer from the set. We develop a heuristic called DOCKS to find a compact UHS, which works in two phases: The first phase is solved optimally, and for the second we propose several efficient heuristics, trading set size for speed and memory. The use of heuristics is motivated by showing the NP-hardness of a closely related problem. We show that DOCKS works well in practice and produces UHSs that are very close to a theoretical lower bound. We present results for various values of k and L and by applying them to real genomes show that UHSs indeed improve over minimizers. In particular, DOCKS uses less than 30% of the 10-mers needed to span the human genome compared to minimizers. The software and computed UHSs are freely available at `github.com/Shamir-Lab/DOCKS/` and `acgt.cs.tau.ac.il/docks/`, respectively.

This study is presented in Chapter 2. It was published as:

Orenstein, Y.*, Pellow, D.*, Marçais, G., Shamir, R., & Kingsford, C. **Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing.** *PLoS Computational Biology* (2017) *13*(10), e1005777 [88].

1.2 Classification of metagenomic sequences

1.2.1 Metagenomic sequencing and the microbiome

A microbiome sample contains microbes (bacteria, viruses, eukaryotes etc.) from a single location or environment that can be sequenced together using high throughput sequencing methods. This *metagenomic sequencing* can reveal the composition of the microbial community from a given environment, known as its *microbiome*. Many different types of environments have been extensively studied, including water sources [124, 64], waste treatment facilities [21], soil [34], animal digestive systems [119], human environments such as homes [39], offices [20] or hospitals [98], and of course human body sites [18, 96, 56, 68, 100, 116].

Different sequencing technologies and protocols can be used to interrogate the microbiome. Sequencing can use high throughput short reads or, more recently, long reads [65]. Whole genome sequencing can resolve individual genomes, and also may facilitate the characterization of genes, functions, strains, mutations etc. that exist in a microbial community. Alternatively, shotgun 16S rRNA sequencing allows for the identification of bacteria present in the sample without the need for assembling full genome sequences. Microbiome studies often examine the microbiome composition or compare composition across individuals, environments, disease states or time points. Metagenomics also enables the discovery and sequencing of new bacterial species (or viruses, archaea, plasmids etc.) that cannot be cultured. This is vital for basic microbiological research and can also have clinical or biotechnological applications. Finally, knowing the composition and genetic makeup of the microbiome allows for a functional analysis of complex microbial communities.

More information about microbiome studies and metagenomic sequencing can be found in the many reviews of the topic. A few example review papers are [121, 38, 23, 82].

1.2.2 Assembling short read sequences

Short read high throughput sequencing experiments generate millions of sequences of length ∼50-200bp, sampled, with sequencing errors, from the DNA sequence of a biological sample. These overlapping reads must then be *assembled* to reconstruct

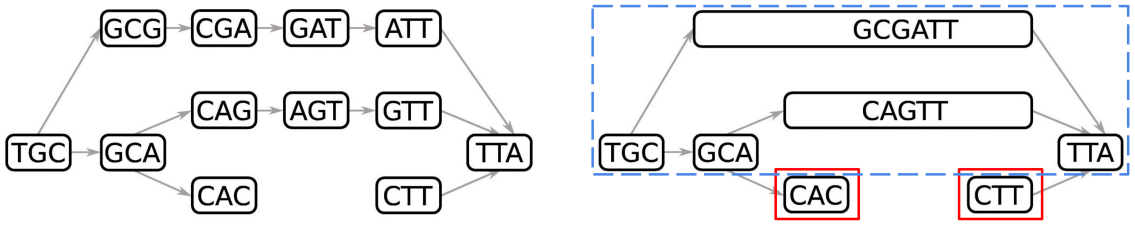


Figure 1.3: **de Bruijn graph compaction and cleaning.** Left: An example order 3 de Bruijn graph. Right: The two non-branching paths have been compacted into a single node. The graph can be cleaned by removing dead-end nodes (red boxes), and collapsing similar parallel paths (blue dashed box) when one is not well-supported by the reads. Figure adapted from: [43].

the genomes that they were sequenced from. Modern short read assemblers use the de Bruijn graph data structure [26] to achieve this [94, 45]. *The de Bruijn graph of order* k of the data consists of a node for every k-mer in the reads and a directed edge from node s to node t if the $k-1$ long suffix of s is the same as the $k-1$ long prefix of t. Paths in the graph represent longer sequences that can be assembled from the reads.

Unambiguous non-branching paths in the de Bruijn graph can be compacted into a single node representing the entire sequence along the path and the graph undergoes cleaning steps to remove or collapse spurious paths (see Figure 1.3). The resulting graph is called the *assembly graph*. Contiguous sequences that can be unambiguously assembled are called *contigs*. Contigs can be joined together (sometimes with gaps between them) in scaffolds to create larger segments of the underlying genome or even full genomic assemblies.

There are a number of common assemblers for assembling bulk isolate samples such as SOAPdenovo [66], Velvet [134] and ABySS [115] and the highly popular SPAdes [9].

Metagenomic assembly is more complicated than isolate assembly since segments of multiple genomes are represented in the assembly graph, and there is a large variation in abundance of the different genomes in the sample, resulting in different read coverage in different parts of the graph. The assembly graph of a microbiome sample will be much larger, more complex, more tangled and more fragmented than for an isolate sample. Commonly used short-read metagenomic assemblers are metaSPAdes [85] and MEGAHIT [59]. Some example review papers of isolate and metage-

nomic assembly are [81, 84, 15] and some of these tools have been benchmarked in: [79, 80, 15].

1.2.3 Sequence classification

Due to the challenges of analyzing metagenomic sequence data, classification of sequences, whether assembled contigs or individual reads, is often an efficient way to achieve tasks that do not require assembly of full genomes. There are a number of tools for taxonomic classification of reads (i.e. according to their family, genus, species or strain of origin). Among these are Kraken [129] and Kraken2 [128], which match k-mers from the read to the lowest common ancestor on a taxonomic tree that contains each k-mer; Clark [90] indexes only target-specific k-mers and classifies reads according to the target with the most specific k-mers hit; Centrifuge [51] uses an FM index of a large genome database, finds maximal exact matches (MEMs) in each read, and then selects taxa with the highest weighted scores across the matches; and Kaiju [78] similarly uses MEMs in a large database of protein sequences. A review of some of these methods can be found in [15] and a benchmark in [130]. Taxonomic read classifiers can provide quantitative taxonomic profiles of the microbiome.

Other methods focus on classifying larger contigs after assembly. For taxonomic classification of longer contigs, it is possible to use local alignment tools to align contigs to reference genomes. Another task is binary or multi-way classification of contigs by type, for example: bacterial origin, viral origin, plasmid origin etc. In the next section I will focus on plasmid classification. VirFinder [101] is a viral contig classifier; PPR-Meta [33], viralVerify [5], and 3CAC [97] are three-way classifiers that classify contigs as chromosomal, viral, or plasmid. Classifiers may be k-mer based, using the frequency distribution of k-mers as features, or they may use other features such as gene content, sequence coverage, and more.

An effective method of classifying contigs is to first bin or cluster them and then identify each bin based on features of all of its contigs. Even when the bin cannot be classified as a known bacterial species, binning allows multiple contigs to be identified as originating from the same organism (referred to as OTUs – operational taxonomic units). There are many metagenomic binning methods that cluster based on sequence composition, coverage in a sample or across multiple samples, gene con-

tent, or a combination of these features. Reviews and benchmarks of metagenomic binning tools can be found in [15, 79, 133].

1.2.4 Plasmid sequence classifiers

In the second paper in this thesis I developed an improved plasmid classifier for metagenomic contigs [92]. In this subsection I will provide background on existing and subsequently developed tools for plasmid sequence classification. (In the next section I will motivate the study of plasmids in more detail.)

The task of plasmid sequence classification is to assign a binary label or probability score to each contig in an assembly classifying it as being of plasmid origin or not. Some classification tools are specifically designed to identify plasmids in isolate assemblies, in our work we focused on identifying plasmid contigs in metagenomic assemblies.

PlasmidFinder [19] identifies BLAST matches to known plasmid replicon sequences in *Enterobacteriaceae*. Gomi et al. [41] used a custom *Klebsiella pneumoniae* plasmid database in Kraken to identify plasmid contigs in *K. pneumoniae* isolates. MOB-recon [103] uses BLAST matches to a curated database of plasmid genes and to known plsamid references to identify plasmids in isolate assembly. Platon [110] looks for enrichment of plasmid or bacterial replicon protein sequences to identify plasmid contigs in isolate.

cBar [139] is an SVM classifier trained on 5-mer frequencies in chromosomal and plasmid sequences from almost 1000 bacterial references. mlplasmids [7] also uses an SVM trained with 5-mer frequencies for classification in *E. faecium*, *K. pneumoniae*, and *E. coli* isolates. PlasmidSeeker [105] identifies plasmids in unassembled isolate reads using a database of k-mers in 10,000 known plasmids. PlasForest [95] is a random forest classifier trained on features of the sequence matches between subsequences of the chromosomes and plasmids of over 10,000 bacterial assemblies and a database of almost 40,000 plasmid reference sequences. RFPlasmid [123] is another random forest classifier that uses 5-mer and marker gene features. plasmidVerify [4] uses a naive Bayes classifier with hmm-profile matches to plasmid genes as the features. PlasX [131], uses a logistic regression model on all gene families in the sequence.

PlasFlow [55] is a neural network based classifier trained on the tf-idf (term

frequency-inverse document frequency, which weights the importance of k-mers in a sequence given a set of sequences) count statistics of different length k-mers in subsequences of almost 10,000 chromosomal and plasmid references. Deeplasmid [2] is another neural network based method trained on 300bp subsequences and sequence features such as GC content, length, gene hits, PFam hits etc. on almost 30,000 plasmids and 40,000 bacterial chromosomes.

The most recent classifiers use assembly graph connections to improve classification. GraphPlas [127] computes similarity in 4-mer composition, coverage, and assembly graph topology to propagate classification labels from confidently classified contigs to other contigs. 3CAC [97] also corrects and propagates plasmid and virus classifications to neighboring contigs in the assembly graph. plASgraph [114] uses a graph neural network encoding the structure of the assembly graph, and contig length, coverage, GC content, k-mer distance from the sample k-mer profile, and node degree features to classify plasmid contigs in isolate samples.

1.2.5 My contribution to plasmid sequence classification

We developed PlasClass, a new plasmid classifier. It uses a set of standard classifiers trained on the most current set of known plasmid sequences for different sequence lengths. We tested PlasClass sequence classification on held-out data and simulations, as well as publicly available bacterial isolates and plasmidome samples and plasmids assembled from metagenomic samples. PlasClass outperforms the state-of-the-art plasmid classification tool on shorter sequences, which constitute the majority of assembly contigs, allowing it to achieve higher F1 scores in classifying sequences from a wide range of datasets. PlasClass also uses significantly less time and memory. PlasClass can be used to easily classify plasmid and bacterial genome sequences in metagenomic or isolate assemblies. It is available under the MIT license from: `https://github.com/Shamir-Lab/PlasClass`.

This study is presented in Chapter 3. It was published as:

Pellow, D., Mizrahi, I., & Shamir, R. **PlasClass improves plasmid sequence classification.** *PLoS Computational Biology* (2020) *16*(4), e1007781 [92].

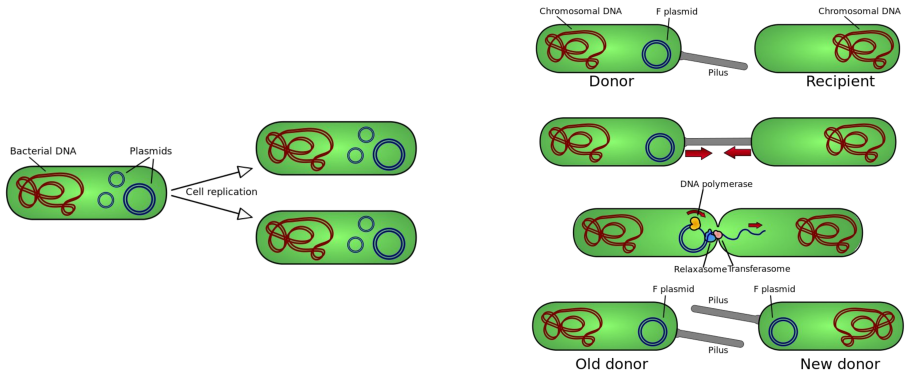


Figure 1.4: **Illustration of plasmid replication and conjugation.** Left: Plasmids are small circular extrachromosomal DNA molecules. The plasmid is replicated along with the host cell DNA. Right: Conjugative plasmids contain genes for pilus construction and transfer, replicating themselves into other host cells.
[Figures adapted from Wikimedia Commons. Left: "Plasmid replication", by Spaully Right: "Overview of bacterial conjugation", by Adenosine.]

1.3 Plasmid reconstruction

1.3.1 Plasmids: biological and clinical motivation

Plasmids are small extrachromosomal DNA molecules in bacterial cells that are able to replicate using the host cell replication machinery, and may be integrated into the host genome. Plasmids are usually circular and typically range in size from around 1kb to hundreds of kilobases. Conjugative plasmids may transfer from one host cell to another, facilitating horizontal gene transfer in bacterial populations. See Figure 1.4 for a visualization. Plasmids are of biological and clinical interest as they can carry antibiotic resistance or other metabolically advantageous genes, and can be responsible for transferring them throughout or across bacterial populations.

Plasmids can be classified according to their replication and transfer genes, and are grouped into incompatibility classes based on them. Reviews of plasmid classifications can be found in [113, 89]. More elaborate classification methods have been proposed more recently [36].

Plasmid biology, ecology, and evolution have been extensively studied. Some recent papers and reviews of the subject for further reference can be found in [104, 50, 131]. A large number of papers also review discovery and surveillance of plasmid mediated virulence and antibiotic resistance, for example [54, 77, 24]. As in the case

of bacterial genomes, metagenomic samples greatly advanced the study of plasmids. Several recent large-scale studies have begun to elucidate the landscape of plasmids based on large sets of metagenomic samples [131, 17, 120].

1.3.2 Plasmid assembly

An important step in characterizing and studying plasmid biology and ecology is the assembly of plasmid sequences. Although for some biological questions using the classification methods described above is sufficient, others require assembly of complete plasmid sequences. In the third paper in this PhD, I developed a metagenomic plasmid assembler for this purpose [93]. In this section I introduce plasmid assembly methods.

Most plasmid assembly methods build on existing assemblers such as the ones described in Chapter 1.2.2. Many of the early plasmid assemblers attempted to assemble plasmids in isolate samples. Plasmid assembly in isolate is itself a difficult task, an early review paper of assembly methods was title "On the (im)possibility of reconstructing plasmids from whole-genome short-read sequencing data" [8]. One option to assemble plasmids in isolate samples is to use a regular assembler, identify plasmid contigs (for example using a plasmid classifier), and manually determine the contig layout. Placnet [58] is a visualization tool that enables this pipeline. Unicycler [126] is an assembly tool that post-processes the assembly graph constructed by SPAdes using graph cleaning, repeat resolution, and bridging steps. It is able to construct complete circular sequences in isolate samples, and can construct complete plasmid sequences. PlasmidSPAdes [3] is a variant of the SPAdes assembler that was modified to assemble plasmids from isolate samples. It creates a "plasmid graph" by removing edges in the assembly graph with coverage close to the median, trimming neighbouring edges that then become disconnected, and removing small non-circular connected components. The idea is that components in the graph with significantly different coverage than the rest of the graph are likely plasmids. HyAsP [83] takes as input a Unicycler assembly graph for an isolate sample; identifies likely plasmid contigs based on gene content, differential GC-content and read depth; and greedily extends a path from likely plasmid contigs to neighboring contigs with plasmid genes and similar depth and GC-content.

Plasmid assembly in metagenomic samples is an inherently much more difficult

task for several reasons: samples are much larger and more complex, the plasmids included in them are only a small fraction of the sample, there are more plasmids, and plasmids from different species may be similar to each other. The result is large and fragmented assembly graphs with plasmid sequences that are not easily identifiable or separable subgraphs. In addition, because of the variation in abundance among organisms in a metagenomic sample, the differential abundance of plasmids cannot be used to identify them. The CAMI 2 benchmark [79] included results on circular elements that demonstrated the difficulty of existing (non-plasmid-specific) metagenomic assemblers in assembling plasmid sequences.

There are a number of metagenomic-specific plasmid assembly tools. Recycler [107] iteratively finds cyclic paths with uniform coverage that are concordant with paired-end read matches in a metagenomic assembly graph and reports them as plasmids. After constructing a plasmid sequence in this way, the cycle is "peeled" out of the assembly graph and the corresponding coverage values are updated. metaplasmidSPAdes [4] is a variant of of the SPAdes assembler for metagenomic plasmids. It iteratively generates smaller and smaller subgraphs of the assembly graph by removing contigs with coverage below a threshold that increases in each iteration. As lower coverage segments of the graph are removed, longer contigs may be constructed in the remaining subgraph. Cyclic contigs are considered as putative plasmids and then verified using the profile of their genetic contents. Domcycle [111] generates cyclic paths in an assembly graph and performs statistical tests on the read coverage and orientation of paired-end reads mapping in and out of the cycle to determine whether they are true plasmid sequences. Yu et al. [131] identify cyclic contigs in the assembly graph and then use PlasX to identify those that are plasmids.

As in the case of metagenomic assembly of bacterial genomes, binning plasmid contigs into individual plasmid bins may be a more effective option than attempting to fully assemble complete plasmid sequences. Plasmid bins could then be reassembled or treated as "plasmid OTUs". Plasmid binning is a more difficult task than classification, which groups all plasmid contigs together. It is also more difficult than metagenomic binning as the plasmid contigs are a small, unidentified fraction of the entire assembly, and thus global binning, aimed at clustering all contigs, will not necessarily perform well at binning individual plasmids. A recent benchmark showed that existing metagenomic binning tools do not successfully bin plasmid contigs [69].

PlasBin [71] is a mixed integer linear programming method for creating plasmid bins in isolate samples. It maximizes plasmid gene density and minimizes deviation from average GC content along a path of selected plasmid contigs in an assembly graph. gplas [6] also bins plasmid contigs in isolate samples by generating walks between plasmid contigs that are consistently covered. A plasmid contig graph is then generated, connecting all contigs that are reachable in the walks into connected components and splitting the components using graph partitioning algorithms.

1.3.3 My contribution to metagenomic plasmid assembly

We developed SCAPP (Sequence Contents-Aware Plasmid Peeler) – an algorithm and tool to assemble plasmid sequences from metagenomic sequencing. SCAPP builds on some key ideas from the Recycler algorithm while improving plasmid assemblies by integrating biological knowledge about plasmids. We compared the performance of SCAPP to Recycler and metaplasmidSPAdes on simulated metagenomes, real human gut microbiome samples, and a human gut plasmidome dataset that we generated. We also created plasmidome and metagenome data from the same cow rumen sample and used the parallel sequencing data to create a novel assessment procedure. Overall, SCAPP outperformed Recycler and metaplasmidSPAdes across this wide range of datasets.

SCAPP is an easy to use Python package that enables the assembly of full plasmid sequences from metagenomic samples. It outperformed existing metagenomic plasmid assemblers in most cases and assembled novel and clinically relevant plasmids in samples we generated such as a human gut plasmidome. SCAPP is open-source software available from: `https://github.com/Shamir-Lab/SCAPP`.

This study is presented in Chapter 4. It was published as:

Pellow, D., Zorea, A., Probst, M., Furman, O., Segal, A., Mizrahi, I., & Shamir, R. **SCAPP: an algorithm for improved plasmid assembly in metagenomes.** *Microbiome* (2021), *9*(1), 1-12 [93].

Chapter 2

Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing

PLOS | COMPUTATIONAL BIOLOGY

RESEARCH ARTICLE

Designing small universal *k*-mer hitting sets for improved analysis of high-throughput sequencing

Yaron Orenstein[1☯], David Pellow[2☯], Guillaume Marçais[3], Ron Shamir[2]*, Carl Kingsford[3]*

1 Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, Massasschusetts, United States of America, **2** Blavatnik School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel, **3** Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, United States of America

☯ These authors contributed equally to this work.
* carlk@cs.cmu.edu (CK); rshamir@cs.tau.ac.il (RS)

OPEN ACCESS

Citation: Orenstein Y, Pellow D, Marçais G, Shamir R, Kingsford C (2017) Designing small universal *k*-mer hitting sets for improved analysis of high-throughput sequencing. PLoS Comput Biol 13(10): e1005777. https://doi.org/10.1371/journal.pcbi.1005777

Editor: Benjamin J. Raphael, Princeton University, UNITED STATES

Received: February 13, 2017

Accepted: September 18, 2017

Published: October 2, 2017

Data Availability Statement: All relevant data are within the paper and its Supporting Information files. The software is available from github.com/Shamir-Lab/DOCKS and k-mer sets are available from acgt.cs.tau.ac.il/docks website.

Funding: RS was supported in part by the Israel Science Foundation as part of the ISF-NSFC joint program 2015-2018. DP was supported in part by a Ph.D. fellowship from the Edmond J. Safra Center for Bioinformatics at Tel-Aviv University and in part by an Israel Ministry of Immigrant

Abstract

With the rapidly increasing volume of deep sequencing data, more efficient algorithms and data structures are needed. Minimizers are a central recent paradigm that has improved various sequence analysis tasks, including hashing for faster read overlap detection, sparse suffix arrays for creating smaller indexes, and Bloom filters for speeding up sequence search. Here, we propose an alternative paradigm that can lead to substantial further improvement in these and other tasks. For integers k and $L > k$, we say that a set of k-mers is a *universal hitting set* (UHS) if every possible L-long sequence must contain a k-mer from the set. We develop a heuristic called DOCKS to find a compact UHS, which works in two phases: The first phase is solved optimally, and for the second we propose several efficient heuristics, trading set size for speed and memory. The use of heuristics is motivated by showing the NP-hardness of a closely related problem. We show that DOCKS works well in practice and produces UHSs that are very close to a theoretical lower bound. We present results for various values of k and L and by applying them to real genomes show that UHSs indeed improve over minimizers. In particular, DOCKS uses less than 30% of the 10-mers needed to span the human genome compared to minimizers. The software and computed UHSs are freely available at github.com/Shamir-Lab/DOCKS/ and acgt.cs.tau.ac.il/docks/, respectively.

Author summary

High-throughput sequencing data has been accumulating at an extreme pace. The need to efficiently analyze and process it has become a critical challenge of the field. Many of the data structures and algorithms for this task rely on k-mer sets (DNA words of length k) to represent the sequences in a dataset. The runtime and memory usage of these highly depend on the size of the k-mer sets used. Thus, a minimum-size k-mer hitting set, namely, a set of k-mers that hit (have non-empty overlap with) all sequences, is desirable.

Absorption fellowship. This research is funded in part by the Gordon and Betty Moore Foundation's Data-Driven Discovery Initiative through Grant GBMF4554 to CK, by the US National Science Foundation (CCF-1256087, CCF-1319998) and by the US National Institutes of Health (R01HG007104). CK received support as an Alfred P. Sloan Research Fellow. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

In this work, we create universal *k*-mer hitting sets that hit any *L*-long sequence. We present several heuristic approaches for constructing such small sets; the approaches vary in the trade-off between the size of the produced set and runtime and memory usage. We show the benefit in practice of using the produced universal *k*-mer hitting sets compared to minimizers and randomly created hitting sets on the human genome.

Introduction

The pace of high-throughput sequencing keeps accelerating as it becomes cheaper and faster and with it the need for faster and more memory efficient genomic analysis methods grows. The NIH Sequence Read Archive, for example, currently contains over 12 petabases of sequence data and is growing at a fast pace. Increased use of sequence-based assays (DNA sequencing, RNA-seq, numerous other "*-seq"s) in research and in clinical settings creates high computational processing burdens. Metagenomic studies generate even larger sequencing datasets. New fundamental computational ideas are essential to manage and analyze these data.

The minimizer approach has been extremely successful in increasing the efficiency of several sequence analysis challenges. Given a sequence of length L, its *minimizer* is the lexicographically smallest k-mer in it [1, 2]. For a sequence S of any length its *minimizer set* is the set of minimizers of every L-long subsequence in S. Hence, every window of length L in S is represented in the set, and a minimizer set for a sequence S constitutes a succinct representation for it. As we discuss below, minimizers have had numerous applications in sequence analysis.

Here, we generalize and improve on the minimizer idea. To avoid dependence on a particular sequence S, we introduce the notion of a universal hitting set. For integers k, L, a set $U_{k,L}$ is called a *universal hitting set* of k-mers (UHS) if every possible sequence of length L must contain at least one k-mer from $U_{k,L}$. The set of all k-mers is a trivial UHS, but it does not provide any useful reduction in computational resources needed. Hence, our main computational problem is:

Problem 1. Given k and L, find a smallest universal hitting set of k-mers.

A small UHS has a variety of applications in speeding up genomic analyses since it can be used where minimizers have been used in the past. For example:

1. **Hashing for read overlapping.** A naïve read overlapper must test $O(n^2)$ pairs of reads to see whether they overlap (where n is the number of reads). If we require an overlap of length L, any pair of reads with such an overlap must share a k-mer from set $U_{k,L}$ in this overlapped region. By bucketing reads into bins according to the universal k-mers they contain, we need only test pairs of reads in the same bucket. The number of buckets is limited by $|U_{k,L}|$.

2. **Sparse suffix arrays.** A sparse suffix array of a string S saves memory by storing an index for only every sth position in S [3]. To query a sparse suffix array for string q, we perform at most s queries starting from indices $0, \ldots, s-1$ in q; one of these queries will intersect a position stored in the suffix array. Using $U_{k,L}$, we can instead store only positions in S that start with a k-mer in $U_{k,L}$. Any query with $|q| \geq L$ must contain one of these selected k-mers and will be matched when searching the suffix array. This approach has been applied with minimizers [4] to good effect.

3. **Bloom filters to speed up sequence search.** Bloom filters have been used to speed up sequence search by storing k-mers present in a read set for quick testing [5, 6]. In current

implementations, all k-mers present in a read set are stored in these filters. If, instead, only the set of k-mers in $U_{k,L}$ is stored, any window of length $\geq L$ is still guaranteed to contain one of these representative queries, potentially reducing the size of Bloom filters that must be maintained.

Minimizers have been used for some of these and similar applications [4, 7–9]. They were originally introduced by Roberts *et al.* [1] for genome assembly. The same idea was introduced independently for plagiarism detection in Schleimer *et al.* [2]. For example, MSP [10] compresses k-mers by hashing them to their 4-mer minimizer to efficiently construct a de Bruijn graph for assembly. SparseAssembler [11] represents the de Bruijn graph using only every g-th k-mer in the sequence (and has also been implemented using minimizers). Kraken [12] uses minimizers to speed up database queries for k-mers during metagenome sequence classification. KMC 2 [8] uses minimizers to cluster subsequences for counting k-mer occurrences. The Locally Consistent Parsing (LCP) [13] algorithm provides the concept of "core substrings" which, like minimizers, are guaranteed to be shared by long enough identical strings. SCALCE [14] uses core substrings to compress DNA sequences.

A small UHS, if it can be found, has a number of advantages over minimizers for these applications:

1. The set of minimizers for a given collection of reads may be as dense as the complete set of k-mers (size $|\Sigma|^k$ for an alphabet Σ), whereas we show that we can often generate UHSs smaller by a factor of nearly k. We also demonstrate on real genomic sequences that the number of UHS k-mers needed to process them is substantially smaller.
2. For any k and L, a set of universal k-mers needs to be computed only once and not recomputed for every dataset.
3. The hash buckets, sparse suffix arrays, and Bloom filters created for different datasets will contain a comparable set of k-mers if they are sampled according to a UHS. This will enable easier comparison and integration of the datasets.
4. One does not need to look at the reads or to build a dataset-specific de Bruijn graph in order to decide which k-mers to use.

Problem 1 can be rephrased as a problem on the complete de Bruijn graph of order k (see Definition 1 below). This is the viewpoint we take for most of this study:

Problem 2. Given a de Bruijn graph D_k of order k and an integer L, find a smallest set of vertices $U_{k,L}$ such that any path in D_k of length $\ell = L - k$ passes through at least one vertex of $U_{k,L}$.

Here and throughout, the length of a path is the number of *edges* in it. We show that the related problem of finding a minimum-size k-mer set that hits every string in a given set $\tilde{S}$ of L-long strings is NP-hard. This problem differs from ours, in that the set $\tilde{S}$ is part of the input. However, the fact that finding a small set of k-mers that hits every sequence in a particular data set is hard further motivates the need for a universal set that can be computed once for any input sequence. Our main contribution is an algorithm called DOCKS that finds a compact set of k-mers that hits any L-long sequence. We also provide several variants of the algorithm, trading-off some solution quality for speed. We show empirically that the produced sets are often close to a theoretical lower bound, implying their near-optimality. Our use of a greedy heuristic is motivated by the fact that finding a minimum-size ℓ-long path cover in a graph G is NP-hard when G is a directed acyclic graph (DAG). We report on the size of the universal k-mer hitting set produced by DOCKS and demonstrate on genomic datasets that we can more uniformly cover sequences with a smaller set of k-mers than is possible using

minimizers. For example, we show that the number of *k*-mers needed to cover the human genome using a UHS is less than one third of that required by minimizers.

The software to compute small UHSs is freely available at github.com/Shamir-Lab/DOCKS/. Universal sets of *k*-mers computed by DOCKS for a range of values of *L* and *k* are freely available at acgt.cs.tau.ac.il/docks/. A preliminary version of this study appeared in [15].

Preliminaries

Throughout this paper, *k* denotes the length of a *k*-mer word, while *L* denotes the length of the long sequences.

Definition 1 (de Bruijn Graph). A *de Bruijn graph* of order *k* over alphabet Σ is a directed graph in which every vertex has an associated label (a string over Σ) of length *k* (*k*-mer) and every edge has an associated label of length $k+1$. There are exactly $|\Sigma|^k$ vertices in a de Bruijn graph, each representing a unique *k*-mer. If an edge (u, v) has label *l*, then the label of *u* must be the *k*-prefix (prefix of length *k*) of *l* and the label of *v* must be the *k*-suffix (suffix of length *k*) of *l*. A *complete* de Bruijn graph contains all possible edges of this type, which represent together all $(k+1)$-mers over Σ.

Every path in a de Bruijn graph represents a sequence. A path $v_0, e_0, v_1, e_1, v_2, \ldots, v_n$ of length *n* spells a sequence *s* of length $n+k$ such that the label of v_i occurs in *s* starting at position *i* for all $0 \leq i \leq n$, and the label of e_i occurs in *s* starting at position *i* for all $0 \leq i \leq n-1$. Note that vertices and edges may repeat in a path.

We define terminology for *k*-mers intersecting sequences over an alphabet Σ:

Definition 2 (hits). We say that *k*-mer *w hits* string *S*, denoted $w \subseteq S$, if *w* appears as a contiguous substring in *S*. *k*-mer set *X hits* string *S* if there exists $w \in X$ s.t. $w \subseteq S$. Define $hit(w, L) = \{S \in \Sigma^L \mid w \subseteq S\}$ for *k*-mer *w* and length *L*, where Σ^L is the set of all *L*-long substrings over alphabet Σ. Define $hit(X, L) = \bigcup_{w \in X} hit(w, L)$.

The universal set of hitting *k*-mers from Problem 1 is then a set $U_{k,L}$ which satisfies $hit(U_{k,L}, L) = \Sigma^L$.

Materials and methods

It is not known how to efficiently find a minimum universal (k, L)-hitting set. As we prove in the Appendix, the problem of finding a minimum (non-universal) *k*-mer set that hits a given set of input sequences is NP-hard (see Appendix, Subsection NP-hardness of MINIMUM (k, L)-HITTING SET in S1 Text). In the face of the hardness result for this related problem, we give below a practical heuristic to find a compact (near-optimal) universal *k*-mer set. This algorithm works on the de Bruijn graph of order *k* in two steps: first it finds and removes a minimum-size *k*-mer set hitting all infinite sequences, and then it finds and removes additional *k*-mers in order to hit all remaining *L*-long sequences. We now describe these two steps in detail.

Finding a minimum *k*-mer set hitting all infinite sequences

The problem of finding a minimum-size *k*-mer set hitting all infinite sequences is known in the literature as finding an unavoidable set of constant length [16]. Note that finite words may avoid the set. Finding a minimum-size unavoidable set for a given *k* can be solved in time polynomial in the output size [16]. The original algorithm is due to Mykkeltveit [17]. Its running time is $O(kM(k))$, where $M(k)$ is the size of the minimum unavoidable set. $M(k)$ converges to $|\Sigma|^k/k$ (an exact formula is given in Eq 11), so the running time is $O(|\Sigma|^k)$.

An unavoidable set of constant length *k* is equivalent to a set of vertices in a complete de Bruijn graph of order *k* whose removal turns it into a directed acyclic graph (DAG). Each

k-mer in the set corresponds to a vertex, and the removal of vertices from every cycle guarantees that no infinite sequence is represented as a path in the graph. This set is known as a *decycling set*.

Hitting remaining length *L* sequences

Unfortunately, finding an unavoidable set is not enough, as there may be L-long sequences that avoid that set. Thus, we need additional k-mers to hit those. If we consider the graph formulation, after removal of a decycling set from the graph we are left with a DAG, which may contain $(L-k)$-long paths representing L-long sequences. We need to remove additional vertices, so that there is no path of length $\ell = L-k$. The problem of finding a minimum-size set of vertices that hit all ℓ-long paths in a general directed acyclic graph is known to be NP-hard, as we review in the Appendix (see Appendix, Subsection NP-hardness of MINIMUM ℓ-PATH COVER IN A DAG in S1 Text). Therefore, we give a heuristic solution.

Our initial algorithm is based on the greedy algorithm for the minimum hitting set [18]. We define the *hitting number* $T(v,\ell)$ of a vertex v to be the number of paths of length ℓ that contain v. The main observation is that we can calculate the hitting number of each vertex efficiently using dynamic programming. The solution is based on calculating the number of paths of length i that terminate at vertex v, and the number of paths of length i that start at vertex v, for all $v \in V$ and $0 \leq i \leq \ell$. Then, the number of ℓ-long paths through v is directly computable from these values by breaking any path into an i-long path ending at v and an $(\ell - i)$-long path starting at v, for all possible values of i. We set $\ell = L-k$ to get the desired hitting number of each vertex.

Specifically, let $G' = (V', E')$ be the directed acyclic graph, after removal of the decycling set. Denote by D and F matrices of size $|V'| \times (\ell+1)$ where $D(v,i)$ is the number of i-long paths in G' starting at vertex v and $F(v,i)$ is the number of i-long paths ending at vertex v.

The calculation of D and F is done recursively as follows:

$$D(v,0) = F(v,0) = 1, \text{for all } v \in V' \tag{1}$$

$$D(v,i) = \sum_{(v,u)\in E'} D(u,i-1) \tag{2}$$

$$F(v,i) = \sum_{(u,v)\in E'} F(u,i-1) \tag{3}$$

To get the number of ℓ-long paths that vertex v participates in, we sum:

$$T(v,\ell) = \sum_{i=0}^{\ell} F(v,i) \cdot D(v,\ell-i) \tag{4}$$

The running time is proportional to the sum of all vertex degrees (which is $\Theta(|E'|)$) times ℓ, giving a running time of $O(|\Sigma|^{k+1} \cdot \ell)$ for $\ell = L-k$.

The DOCKS algorithm

The full algorithm combines the two steps. First, we find a decycling set in a complete de Bruijn graph of order k and remove it from the graph, obtaining a DAG. Then, we repeatedly remove a vertex v with the largest hitting number $T(v,\ell)$ until there are no ℓ-long paths, recomputing $T(u,\ell)$ for all remaining vertices u after each removal. This is summarized below (Algorithm 1).

Algorithm 1 DOCKS: Find a compact *k*-mer set hitting all *L*-long sequences

1: Generate a complete de Bruijn graph G of order k, set $\ell = L - k$.
2: Find a decycling vertex set X using Mykkeltveit's algorithm.
3: Remove all vertices in X from graph G, resulting in G'.
4: **while** there are still paths of length ℓ **do**
5: Calculate $D(v, i)$ and $F(v, i)$ for each vertex v and $0 \leq i \leq \ell$.
6: Calculate $T(v, \ell)$ for each vertex v.
7: Remove a vertex with maximum hitting number from G', and add it to set X.
8: **end while**
9: Output set X.

Finding the decycling set takes $O(|\Sigma|^k)$. In the second phase, each iteration calculates the hitting number of all vertices in time $O(|\Sigma|^{k+1}\ell)$. The number of iterations is $1 + p$, where p is the number of vertices removed. Thus, the total running time is dominated by steps 4–8 and is $O((1 + p)|\Sigma|^{k+1}\ell)$.

The exponential dependence of DOCKS on k limits the range of k to which it can be applied (see Results, Subsection DOCKS). This motivates us to develop two variants that trade larger solution sizes for faster running times in the different heuristics described next.

The DOCKSany algorithms

In order to extend the range of k, L values beyond what DOCKS can compute in reasonable times, we develop a faster heuristic that may produce cruder solutions. Instead of calculating the number of ℓ-long paths through each vertex, we consider *all* paths through each vertex. This number, denoted by $T(v)$, can be calculated more quickly and serve as an estimate of $T(v, \ell)$. We call this heuristic DOCKSany (Algorithm 2).

DOCKSany has the same structure as DOCKS, but with one difference: it removes a node v with maximum $T(v)$ in each iteration. To compute $T(v)$ for all v, the vertices in the current graph $G' = (V', E')$ are first sorted in topological order $v_1 \leq \ldots \leq v_n$. Define $F(v)$ as the number of paths ending at v. The vertices are visited in topological order and the incoming edges into v are used to compute:

$$F(v) = 1 + \sum_{(u,v)\in E'} F(u) \tag{5}$$

Similarly, $D(v)$, the number of paths starting at v is computed by visiting the vertices in reverse topological order and computing.

$$D(v) = 1 + \sum_{(v,u)\in E'} D(u) \tag{6}$$

$T(v)$ is then calculated for all vertices as:

$$T(v) = F(v) \cdot D(v). \tag{7}$$

A vertex v with maximum $T(v)$ is removed, G' is updated, and the process is repeated until there are no paths of length ℓ in the graph.

Algorithm 2 DOCKSany: A faster heuristic for a compact *k*-mer set hitting all *L*-long sequences

1: Generate a complete de Bruijn graph G of order k, set $\ell = L - k$.
2: Find a decycling vertex set X using Mykkeltveit's algorithm.
3: Remove all vertices in X from graph G, resulting in G'.
4: **while** there are still paths of length ℓ **do**
5: Calculate $D(v)$ and $F(v)$ at each vertex v.
6: Calculate the number $T(v)$ of paths passing through each vertex v.
7: Remove a vertex v with maximum $T(v)$ from G', and add it to set X.
8: **end while**
9: Output set X.

Computing $D(v)$ for all v requires visiting each edge in the graph once, and hence takes $O(|\Sigma|^{k+1})$. The time for computing $F(v)$ for all v is the same. Hence, T is computable in $O(|\Sigma|^{k+1})$ time. Computing the longest path in a DAG (step 4) also requires $O(|\Sigma|^{k+1})$. If p vertices are removed, then the total runtime for this algorithm is $O((1+p)|\Sigma|^{k+1})$, a factor of $\Theta(\ell)$ faster than the DOCKS algorithm. The space complexity is also smaller, $O(|\Sigma|^{k+1})$ vs. $O(\ell|\Sigma|^{k+1})$ for DOCKS.

In addition to shorter runtimes and decreased memory usage, this heuristic offers one more advantage over the original DOCKS algorithm. The vertex removal choice is independent of L. The value of L only determines when the algorithm terminates. Thus, hitting sets for all values of L or larger can be computed in one run. This is in contrast with DOCKS, in which the hitting number of each vertex depends on L, and so DOCKS must be run for each desired value of L.

Finally, in order to calculate the hitting set for even larger k, we can further speed up DOCKSany as follows. In the DOCKSanyX heuristic, the top X vertices, ranked by the hitting number $T(v)$, are removed (in step 7) in each iteration. This can shorten the running time of each iteration by a factor of X, but may produce larger hitting set solutions.

An integer linear programming (ILP) formulation

To investigate whether optimal solutions can be found practically, we formulate the problem of the minimal universal k-mer hitting set as an integer linear program (ILP). In the ILP formulation there are $|\Sigma|^k$ binary variables x_i representing whether vertex i is in the solution hitting set. There are also $|\Sigma|^k$ variables L_i representing an upper bound on the number of edges in the longest path ending at vertex i. The constraints on L_i guarantee that the vertices chosen remove all ℓ-long paths ($\ell = L - k$) from the graph. The ILP is defined as follows:

$$\begin{aligned} \text{minimize}: \quad & \sum_{i=1}^{|\Sigma|^k} x_i, & & \\ \text{subject to}: \quad & x_i \in \{0,1\}, & & 1 \le i \le |\Sigma|^k \\ & 0 \le L_i \le \ell - 1, & & 1 \le i \le |\Sigma|^k \end{aligned} \tag{8}$$

$$L_v \ge 1 + L_u - \ell x_v, \qquad (u,v) \in E \tag{9}$$

Here E contains all $|\Sigma|^{k+1}$ possible edges. The constraint on edge (u, v) requires that if v is not in the set then $L_v \geq 1 + L_u$. The validity of this formulation is proven in the Appendix (see Appendix, Subsection Validity of the ILP formulation in S1 Text).

The number of variables and constraints grows exponentially in k, making it hard to use for $k > 7$. However, the ILP solver can start from a feasible solution produced by one of the DOCKS algorithms and improve that solution for a limited set time.

Handling larger *k*

The DOCKS variants described above have exponential dependence in k in both runtime and memory usage. Hence, the range of k values to which they can be applied is limited. To extend this range, we present below a procedure to construct a universal k-mer hitting set by extending UHSs computed for smaller k values. Given a set $U_{K,\,L}$ and integer j, we can construct set $U_{k+j,\,L+j}$ by concatenating all possible j-mers over Σ to each k-mer in $U_{k,\,L}$. Formally,

$$U_{k+j,L+j} = \{w \cdot x \mid w \in U_{k,L}, x \in \Sigma^j\} \tag{10}$$

To see that $U_{k+j,\,L+j}$ is a universal $(k + j)$-mer hitting set, denote by S an $(L + j)$-long sequence. By definition, there must be at least one k-mer $w \in U_{k,\,L}$ that hits S's L-long prefix. $U_{k+j,\,L+j}$ contains all $(k + j)$-mers $w \cdot x$, where x is any j-mer. Thus, it must contain a $(k + j)$-mer that hits S.

For example, by appending all possible 10-mers to each 10-mer in $U_{10,20}$ we obtain $U_{20,30}$. The size of the set $U_{10,20}$ is $|U_{10,20}| = c \cdot dec_{10}$, where $dec_{10} \approx \frac{4^{10}}{10}$ is the size of a minimum decycling set for $k = 10$ (Eq 11). Here $c \geq 1$ is the approximation factor obtained by the UHS. Then, the size of $U_{20,30}$ is $|U_{20,30}| = |U_{10,20}| \cdot 4^{10} = c \cdot dec_{10} \cdot 4^{10} \approx c \cdot \frac{4^{20}}{10} = 2c \cdot \frac{4^{20}}{20}$ by this construction. This is approximately $|U_{20,30}| \approx 2c \cdot dec_{20}$, i.e. the approximation factor doubled.

Results

A theoretical lower bound for $|U_{k,\,L}|$

For a given k-mer w, its *conjugacy class* is the set of k-mers obtained by rotation of w. Conjugacy classes form cycles in the de Bruijn graph and form a partition of the k-mers. The number of conjugacy classes over all k-mers is given by [16]:

$$C(|\Sigma|, k) = \sum_{i=1}^{k} |\Sigma|^{\gcd(i,k)}/k. \tag{11}$$

A decycling set necessarily contains a k-mer from each conjugacy class. Golomb's conjecture, proved by Mykkeltveit [17], states that the smallest decycling set has cardinality $C(|\Sigma|, k)$. Consequently, a minimum hitting set $U_{k,\,L}$ has a size $\geq C(|\Sigma|, k) \geq |\Sigma|^k/k$.

Table 1 reports L_{max}, the length of the longest sequence in a complete de Bruijn graph after a minimum decycling set computed using Mykkeltveit's algorithm is removed, for $k = 2$ to 14. For this range of k, the length of sequences avoiding the decycling set can theoretically be appropriate for long-read sequencing technologies, such as PacBio [19] and Nanopore [20], which produce reads of length $L > 1000$. Such long reads are all hit by a decycling set

Table 1. Length of longest sequence avoiding an unavoidable set for different values of *k*. For each value k, a minimum decycling set was removed from a complete de Bruijn graph, and the length L_{max} of the longest sequence, represented as a longest path, was calculated.

k	2	3	4	5	6	7	8	9	10	11	12	13	14
L_{max}	5	11	20	45	70	117	148	239	311	413	570	697	931

https://doi.org/10.1371/journal.pcbi.1005777.t001

according to Table 1 for $k \leq 14$ (although a shorter window size may be needed to overcome sequencing errors). However, many short reads can avoid the decycling set. Additional *k*-mers must be selected to obtain a hitting set for shorter sequences. Note that different minimum decycling sets may result in different lengths of the longest path in the remaining DAG. Mykkeltveit's approach is different from that of Champarnaud et al. (2004), and the former has an advantage in producing solutions with shorter longest paths [16].

DOCKS

We implemented and ran DOCKS over a range of k and L: $5 \leq k \leq 10$ and $20 \leq L \leq 200$, in increments of 10. The values of k are typical lengths for minimizers, and the L values are typical lengths of short reads. Note that in some applications, like KMC 2 [8] and Kraken [12], the length of the window used (denoted by k there) corresponds to our L parameter, and the length of the minimizers (m in KMC 2) corresponds to our k parameter.

The results are summarized in Fig 1. As expected, the fraction of *k*-mers included in the solution set decreases with L. It is easier to hit longer sequences as they contain more *k*-mers. In addition, running times and memory usage increase exponentially with k. For $k = 10$, DOCKS terminated after more than 2.5 hours and used more than 1 GB of memory. For $k = 11$ and $L = 20$ running time was 128 hours. Hence, DOCKS runtime would be prohibitively long for larger values of k. Running times were benchmarked on a single CPU of a 20-CPU Intel Xeon E5-2650 (2.3GHz) machine with 384GB 2133MHz RAM.

Fig 1A also shows the size of the decycling set for each k. For $k = 10$ and $L = 20$ the number of added *k*-mers roughly equals the size of the decycling set, while for $k = 5$ and $L = 20$ it is only 20% larger. For all values of k, the ratio improves as L grows. We also compared DOCKS to a pure greedy algorithm that repeatedly removes a vertex with a maximum hitting number, without removing a decycling set first. For almost all combinations of (k, L) the size of the produced set, runtime and memory of the greedy algorithm were far greater than those of DOCKS (see Fig A in S1 Text). In particular, the greedy algorithm's runtime was greater by a factor of more than 1000 for $k = 8$ (taking days compared to minutes), and it increased with L, as opposed to DOCKS's runtime, which decreased with L.

DOCKSany

We ran DOCKSany for $5 \leq k \leq 11$ and $20 \leq L \leq 200$. The results for $k = 10$ are shown in Fig 2 and the full results are in S1 Table and visualized in Fig B in S1 Text. In comparison to DOCKS (see Fig C in S1 Text), the produced sets are larger, especially for smaller values of L, and that gap grows with k: from 10% for $k = 5$ to 60% larger for $k = 10$. Set sizes of DOCKS and DOCKSany are closer as L increases and both approach the size of the decycling set. In terms of running time, on the other hand, we see a great benefit in using DOCKSany as runtimes decrease to a small fraction of the DOCKS running times for the larger values of k. We also see reduced memory usage for larger values of k and L (see the table in S1 Table). Still, DOCKSany becomes impractical for $k \geq 13$ (runtime for $k = 12$, $L = 20$ was 45 days), so we turn to another heuristic to increase runtime on the expense of larger set sizes.

DOCKSanyX

We tested the performance of DOCKSanyX for $k = 10$, $20 \leq L \leq 200$ and $X = 5^i$ for $0 \leq i \leq 5$ (Fig D in S1 Text). As expected, the generated set sizes increase with X, but the differences are very small for $X \leq 125$. On the other hand, the running time improves dramatically as X increases and the memory usage also improves with X, albeit not as dramatically (see S1 Table). Fig 2 compares the sizes of the sets generated by DOCKS, DOCKSany, and

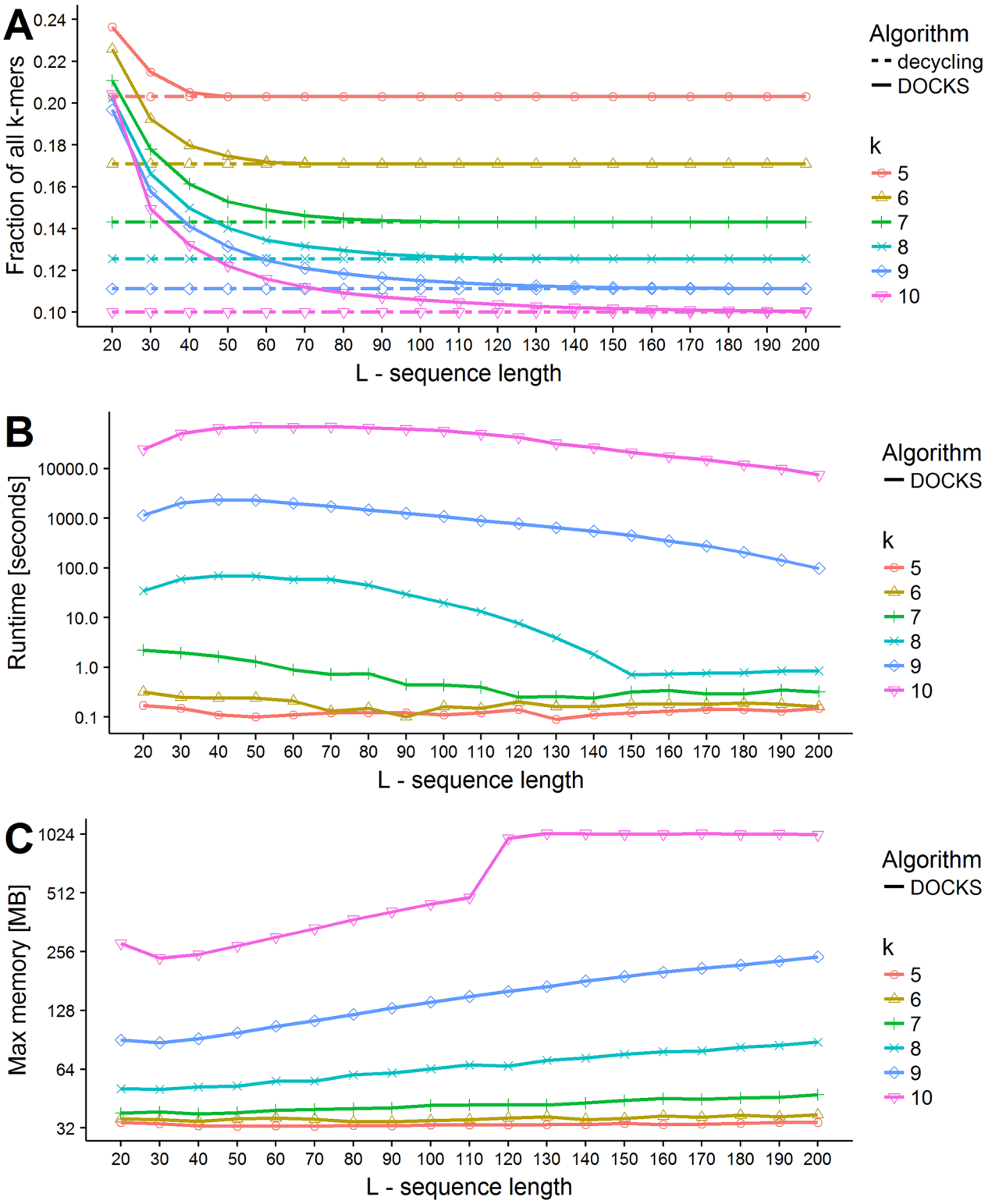


Fig 1. Performance of DOCKS. For different combinations of k and L we ran DOCKS over the DNA alphabet. (A) Set sizes. The results are shown as a fraction of the total number of k-mers $|\Sigma|^k$. The broken lines show the decycling set size for each k. (B) Running time in seconds. Note that y-axis is in log scale. (C) Maximum memory usage in megabytes. Note that y-axis is in log scale.

https://doi.org/10.1371/journal.pcbi.1005777.g001

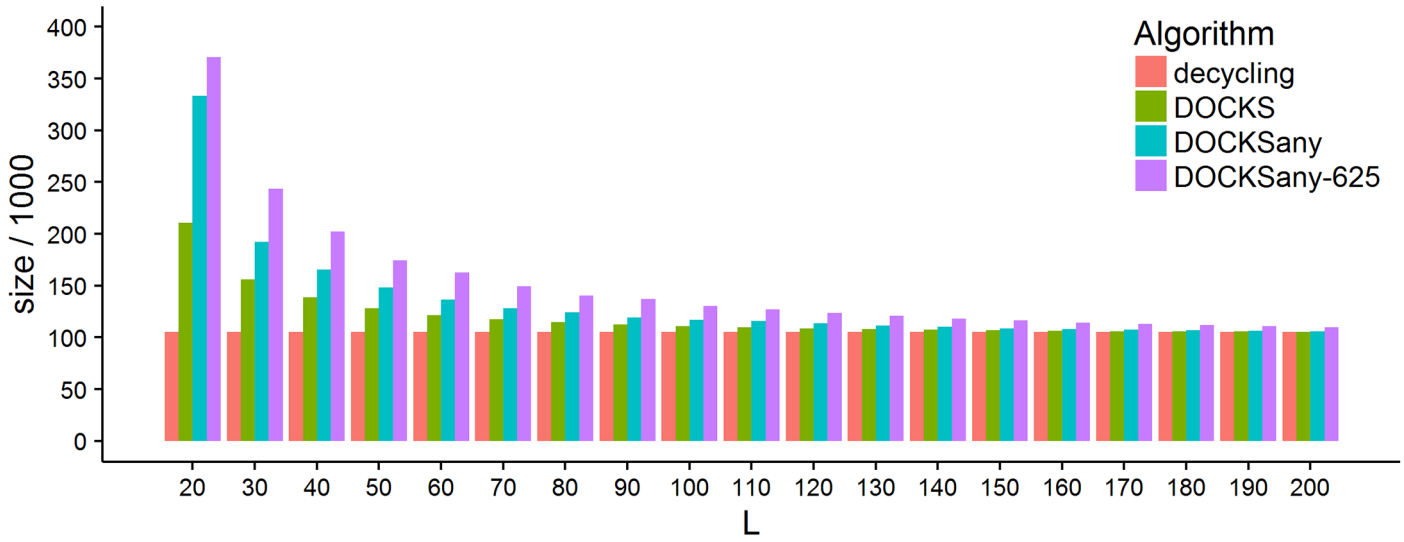


Fig 2. Comparison of the sizes of the universal sets generated by the different heuristics. The histogram shows the size of the universal sets generated by DOCKS, DOCKSany, and DOCKSanyX with $X = 625$. The results are for $k = 10$ and $20 \leq L \leq 200$. The size of the decycling set is provided as a lower bound for comparison.

https://doi.org/10.1371/journal.pcbi.1005777.g002

DOCKSanyX (for $X = 625$). Remarkably, for $k = 10$, the size of the solution is similar to that of DOCKSany while there is a factor of $> 100 \times$ speedup. The results, runtime and memory usage of DOCKSanyX are in S1 Table and visualized in Fig E in S1 Text.

ILP solutions

We solved the ILP using Gurobi 6.5.2 [21] for $5 \leq k \leq 10$ with $20 \leq L \leq 200$. To save time, we set the starting feasible solution to be the DOCKS solution. We let the solver run for up to one day for each k and L. This did not necessarily produce an optimal solution to the ILP, although the solver was often able to improve on the starting DOCKS solution. In Fig 3, we show the

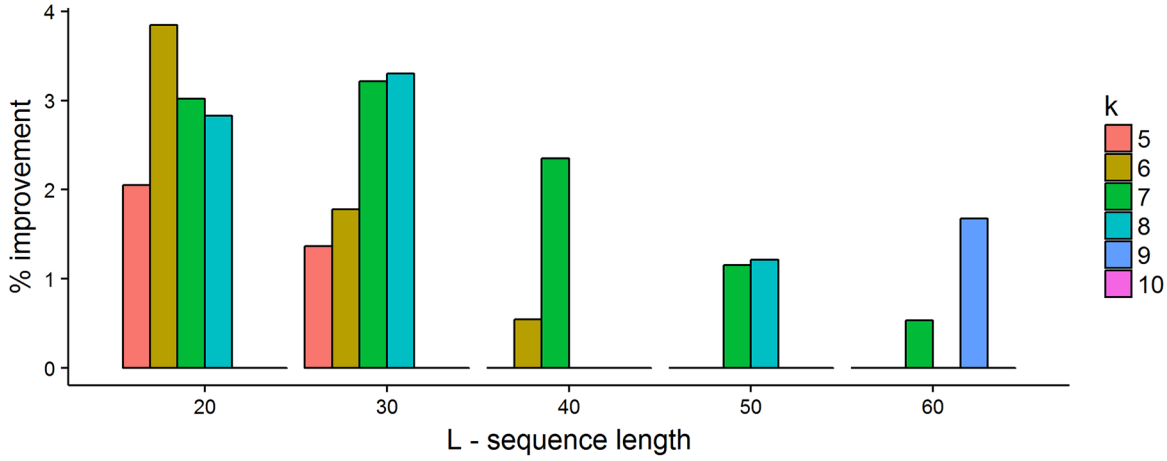


Fig 3. Performance of ILP solver compared to DOCKS. For each combination of $5 \leq k \leq 10$ and $20 \leq L \leq 200$ we ran the ILP solver for up to 24 hours starting from a DOCKS feasible solution. The histograms show the percent improvement of the *k*-mer set size generated by the ILP solver compared to DOCKS. For $L > 60$ and all tested values of k, the improvement was <1%.

https://doi.org/10.1371/journal.pcbi.1005777.g003

Table 2. The number of 10-mers needed to hit all 30-long sequences in four genomes: Two bacterial genomes *A. tropicalis*, *C. crescentus*, the worm *C. elegans* and a mammal genome, *H. sapiens*. The genome sizes are quoted after removing all *N*s and ambiguous codes. We tested three algorithms: minimizers picking the lexicographically smallest 10-mer, minimizer picking the first in a random *k*-mer ordering, and selection using the set produced by DOCKS. In case of multiple DOCKS-selected 10-mers in the 30-long window, the lexicographically smallest was chosen. *# mers* is the number of distinct 10-mers selected, and *avg. dist.* is the average distance between two selected 10-mers.

Species	Genome size (Mbp)	Method	# mers (thousands)	avg. dist.
A. tropicalis	0.393	lexicographic	32.9	9.48
		randomized	28.0	11.0
		DOCKS	23.7	12.4
C. crescentus	4	lexicographic	114.0	10.2
		randomized	89.6	11.0
		DOCKS	66.0	12.4
C. elegans	100	lexicographic	286.0	8.83
		randomized	277.0	11.0
		DOCKS	145.0	12.4
H. sapiens	2900	lexicographic	543.0	9.13
		randomized	389.0	10.9
		DOCKS	154.0	12.1

https://doi.org/10.1371/journal.pcbi.1005777.t002

improvement in the solution set size obtained by the ILP over the DOCKS solution. We can see that using the ILP solver leads to minor improvements over the DOCKS solution (0-4%), especially for small k. Improvements diminish as L increases, since the set sizes approach the theoretical lower bound, i.e., the size of the minimum decycling set. Letting the ILP solver run for longer times may provide further improvements for small values of L.

Comparison to minimizers on several genomes

The minimizer algorithm [1] selects the lexicographically smallest k-mer in each window of w consecutive k-mers in order to reduce storage size for sequence comparison. We can improve the minimizers algorithm by choosing the lexicographically smallest k-mer that is in the DOCKS set for the corresponding k and L parameters (i.e. $L = k + w - 1$). Such a k-mer is guaranteed to exist, as by construction, every window of length w contains a k-mer in the UHS. We ran the minimizer selection algorithm and DOCKS-based selection on four different genomes, using $k = 10$ and $L = 30$: the entire human reference genome (GRCh38), the bacteria *A. tropicalis* strain NBRC 16470, and *C. crescentus* strain CB15, the worm C. elegans assembly WBcel235. For comparison, we also included the results when using the minimizer according to a random ordering of the k-mers, instead of lexicographic. This random ordering typically improves over minimizers since it avoids the problem of always selecting the common poly-A homopolymer.

Table 2 shows that DOCKS selects far fewer k-mers and those k-mers are more widely spread apart in the sequence. The advantage of DOCKS grows as the sequence length increases, having a size $\approx$ 85% of the next-best method for the small bacterial genome, $\approx$ 50% for the larger *C. elegans* genome, and only $\approx$ 40% for the human genome.

Discussion

We presented the DOCKS algorithm, which generates a compact set of k-mers that together hit all L-long DNA sequences. Such compact sets have many applications in sequence analysis, including space efficient data structures and large-scale sequence analysis. We tested the sets

produced by our algorithm in an application that requires finding a small set of 10-mers hitting all 30-long words in the input genomes. Compared to minimizers, the current state of the art, our sets were almost 2.5-3.5 times smaller for the human genome. We could produce sets for the range of $k = 5$ to 10 and $L = 20$ to 200, and the results show that for $L > 100$ the size of the solution is quite close to the theoretical lower bound. We expect the sets produced by our approach to be useful and improving a variety of biological applications that require complex analysis of numerous sequences.

We see the benefit of our compact UHSs in many data structures and algorithms that analyze high-throughput sequencing data. For example, we expect that binning-based *k*-mer counting applications, such as KMC 2 [8], can reduce the number of bins, and thus the number of disk accesses, using universal *k*-mer hitting sets. Analyses that rely on *k*-mer counting, such as metagenomic binning as implemented in Kraken [12], will also see improved computational resource usage. The minimizer idea has been widely deployed, and universal hitting *k*-mers can typically be used as a drop-in replacement, improving computational performance.

The good performance of the algorithms can be attributed to their two phase approach. In the first phase we optimally and rapidly remove a minimum-size set that hits all infinite sequences, which also takes care of many *L*-long sequences. In the second phase we greedily remove *k*-mers that hit remaining *L*-long sequences. Overall efficiency is primarily due to the first phase, which runs in time $O(k)$ times the size of the output. In the second phase dynamic programming is used, providing running time polynomial in the output size.

We developed two additional variants of DOCKS that reduce the runtime and memory usage at the price of increasing the size of the set created. DOCKS can provide a solution for $k = 10$, DOCKSany for $k = 11$, and the fastest variant, DOCKSanyX for $k = 13$ (with $X = 10000$) with $L = 200$, within a day. Note that all heuristics are bound to hit a limit since their runtime depends exponentially on *k*. This is an inherent property of the problem and its output size. Still, we manage to increase *k* by one or two using each heuristic. In partial remedy, we also proposed a construction that can push that limit further at the expense of solution size.

Our approaches are heuristic in nature. This is not surprising, since as we show, the problem of finding a minimum (k, L)-hitting set for a given set of sequences is NP-hard. Moreover, even after removing an optimal decycling set, one needs to solve the problem of finding a minimum vertex set that hits all *L*-long sequences in a directed acyclic graph, which is NP-hard. Hence, DOCKS usually produces sub-optimal solutions. For example, for $k = 4$ and $L = 10$ the optimal solution obtained by solving an ILP formulation had size 89, compared to 91 produced by DOCKS. In fact, our tests show that if further reduction to the hitting set size is needed, starting from the DOCKS solution and improving it using ILP is a good strategy, at least for small values of *k*.

Our study raises several open problems. First, is there a characterization for a minimum universal (k, L)-hitting set similar to the characterization of decycling sets by Mykkeltveit [17]? That is, does there exist an algorithm polynomial in *k* and *L* that can check if a *k*-mer belongs to a particular universal (k, L)-hitting set. The fact that MINIMUM (k, L)-HITTING SET on a given set of input sequences is NP-hard still leaves the universal case open. A related question is whether one can find an algorithm that generates an optimal (universal) (k, L)-hitting set while requiring work polynomial in the output set size. This is particularly interesting for the universal case, where the input is only the values *k* and *L* and the output size is $> |\Sigma|^k/k$. Second, is the problem of minimum ℓ-path cover in a DAG *G* polynomial when *G* is a subgraph of a de Bruijn graph? We know it is hard for a general DAG, but the specific structure of de Bruijn graphs may make the problem easier. Third, the bottleneck to DOCKS running time is the second phase, which currently re-calculates the vertex hitting numbers on each iteration. Can one find a dynamic algorithm that updates these numbers more efficiently after the

removal of one vertex? Fourth, is there a tight upper bound on the number p of vertices that will be removed by the greedy heuristic? Fifth, can we give an upper bound or a tighter lower bound on the size of $U_{k,\,L}$?

Conclusion

We demonstrated the ability of DOCKS to generate compact sets of k-mers that hit all L-long sequences. These k-mer sets can be generated once for any desired value of $k \leq 13$ and L and then readily used for many different purposes. For example, we produced a set of only 700 6-mers out of a total of 4096 that hits every sequence longer than 70 bases—a typical read length for many sequencing experiments—enabling efficient binning of reads. Our compact sets can improve many of the applications that currently use minimizers, as we showed that they are both smaller and more sparsely distributed across genome sequences.

Supporting information

S1 Table. Set size, running time and memory usage of DOCKS, DOCKSany, DOCKSanyX, and the greedy algorithm for the hitting set problem. The table contains solution set size, time in seconds and memory in KB for DOCKS, DOCKSany, DOCKSanyX and the greedy approach algorithms. Note that the reported times are for individual runs of each (k, L) pair, but the sets for all longer L values are computed when computing the (k, $L = 20$) set with DOCKSany or DOCKSanyX and the runtime can be amortized across all of these calculations.
(XLSX)

S1 Text. Supplementary figures and theoretical proofs.
(PDF)

Acknowledgments

Part of this work was done while Y.O., R.S. and C.K. were visiting the Simons Institute for the Theory of Computing.

Author Contributions

Conceptualization: Guillaume Marçais.

Data curation: Guillaume Marçais.

Funding acquisition: Ron Shamir, Carl Kingsford.

Methodology: Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, Carl Kingsford.

Project administration: Yaron Orenstein, David Pellow, Ron Shamir, Carl Kingsford.

Resources: Ron Shamir, Carl Kingsford.

Software: Yaron Orenstein.

Supervision: Ron Shamir, Carl Kingsford.

Validation: Yaron Orenstein, David Pellow, Guillaume Marçais.

Visualization: Yaron Orenstein.

Writing – original draft: Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, Carl Kingsford.

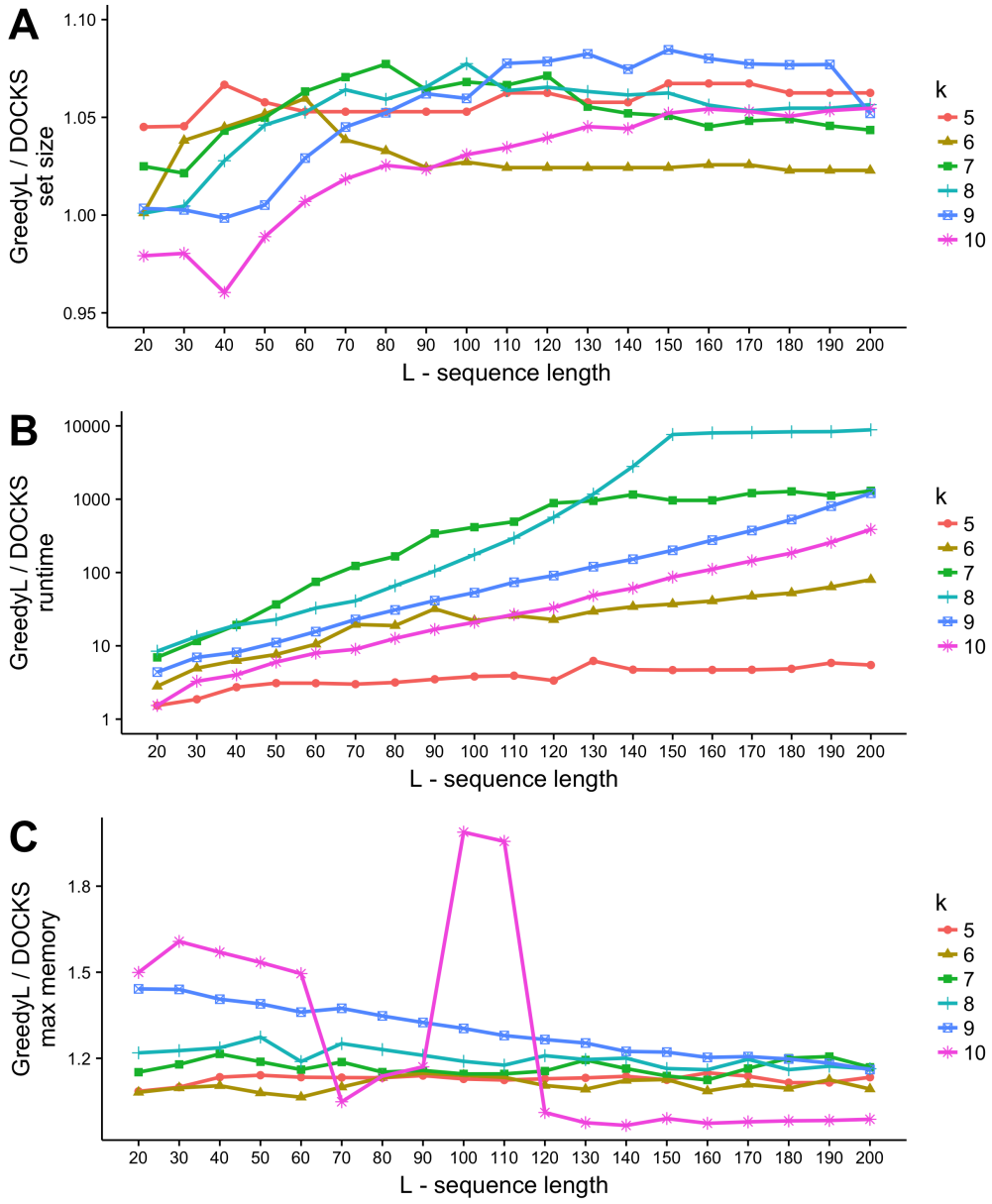
Writing – review & editing: Yaron Orenstein, David Pellow, Guillaume Marçais, Ron Shamir, Carl Kingsford.

References

1. Roberts M, Hayes W, Hunt BR, Mount SM, Yorke JA. Reducing storage requirements for biological sequence comparison. Bioinformatics. 2004; 20(18):3363–3369. https://doi.org/10.1093/bioinformatics/bth408 PMID: 15256412
2. Schleimer S, Wilkerson DS, Aiken A. Winnowing: Local Algorithms for Document Fingerprinting. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. SIGMOD'03. New York, NY, USA: ACM; 2003. p. 76–85. Available from: http://doi.acm.org/10.1145/872757.872770.
3. Karkkainen J, Ukkonen E. Sparse Suffix Trees. In: Computing and Combinatorics: 2nd Annual International Conference, COCOON'96. vol. 2. Springer; 1996. p. 219–230.
4. Grabowski S, Raniszewski M. Sampling the Suffix Array with Minimizers. In: Proceedings of the 22nd International Symposium on String Processing and Information Retrieval. vol. 9309. Springer-Verlag New York, Inc.; 2015. p. 287–298.
5. Solomon B, Kingsford C. Fast search of thousands of short-read sequencing experiments. Nature Biotech. 2016 Mar; 34(3):300–302. https://doi.org/10.1038/nbt.3442
6. Solomon B, Kingsford C. Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees. bioRxiv. 2016; Available from: http://biorxiv.org/content/early/2016/12/02/086561.
7. Movahedi NS, Forouzmand E, Chitsaz H. De novo co-assembly of bacterial genomes from multiple single cells. In: 2012 IEEE International Conference on Bioinformatics and Biomedicine (BIBM); 2012. p. 1–5.
8. Deorowicz S, Kokot M, Grabowski S, Debudaj-Grabysz A. KMC 2: fast and resource-frugal k-mer counting. Bioinformatics. 2015 May; 31(10):1569–1576. Available from: http://bioinformatics.oxfordjournals.org/content/31/10/1569. PMID: 25609798
9. Chikhi R, Limasset A, Jackman S, Simpson JT, Medvedev P. On the representation of de Bruijn graphs. Journal of Computational Biology. 2015 Jan; 22(5):336–352. Available from: http://online.liebertpub.com/doi/abs/10.1089/cmb.2014.0160. PMID: 25629448
10. Li Y, Kamousi P, Han F, Yang S, Yan X, Suri S. Memory efficient minimum substring partitioning. In: Proceedings of the VLDB Endowment. vol. 6. VLDB Endowment; 2013. p. 169–180.
11. Ye C, Ma ZS, Cannon CH, Pop M, Douglas WY. Exploiting sparseness in de novo genome assembly. BMC Bioinformatics. 2012; 13(6):S1. https://doi.org/10.1186/1471-2105-13-S6-S1 PMID: 22537038
12. Wood DE, Salzberg SL. Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biology. 2014; 15(3):R46. https://doi.org/10.1186/gb-2014-15-3-r46 PMID: 24580807
13. Sahinalp SC, Vishkin U. Efficient approximate and dynamic matching of patterns using a labeling paradigm. In: 37th Annual Symposium on Foundations of Computer Science, Proceedings; 1996. p. 320–328.
14. Hach F, Numanagić I, Alkan C, Sahinalp SC. SCALCE: boosting sequence compression algorithms using locally consistent encoding. Bioinformatics. 2012 Dec; 28(23):3051–3057. Available from: http://bioinformatics.oxfordjournals.org/content/28/23/3051 PMID: 23047557
15. Orenstein Y, Pellow D, Marçais G, Shamir R, Kingsford C. Compact universal k-mer hitting sets. In: International Workshop on Algorithms in Bioinformatics. vol. 9838. Springer; 2016. p. 257–268.
16. Champarnaud JM, Hansel G, Perrin D. Unavoidable sets of constant length. International Journal of Algebra and Computation. 2004; 14(02):241–251. https://doi.org/10.1142/S0218196704001700
17. Mykkeltveit J. A proof of Golomb's conjecture for the de Bruijn graph. Journal of Combinatorial Theory, Series B. 1972 Aug; 13(1):40–45. Available from: http://www.sciencedirect.com/science/article/pii/0095895672900068.
18. Chvatal V. A greedy heuristic for the set-covering problem. Mathematics of Operations Research. 1979; 4(3):233–235. https://doi.org/10.1287/moor.4.3.233
19. Rhoads A, Au KF. PacBio sequencing and its applications. Genomics, proteomics & bioinformatics. 2015; 13(5):278–289. https://doi.org/10.1016/j.gpb.2015.08.002
20. Branton D, Deamer DW, Marziali A, Bayley H, Benner SA, Butler T, et al. The potential and challenges of nanopore sequencing. Nature biotechnology. 2008; 26(10):1146–1153. https://doi.org/10.1038/nbt.1495 PMID: 18846088
21. Gurobi Optimization, Inc. Gurobi Optimizer Reference Manual; 2016. Available from: http://www.gurobi.com.

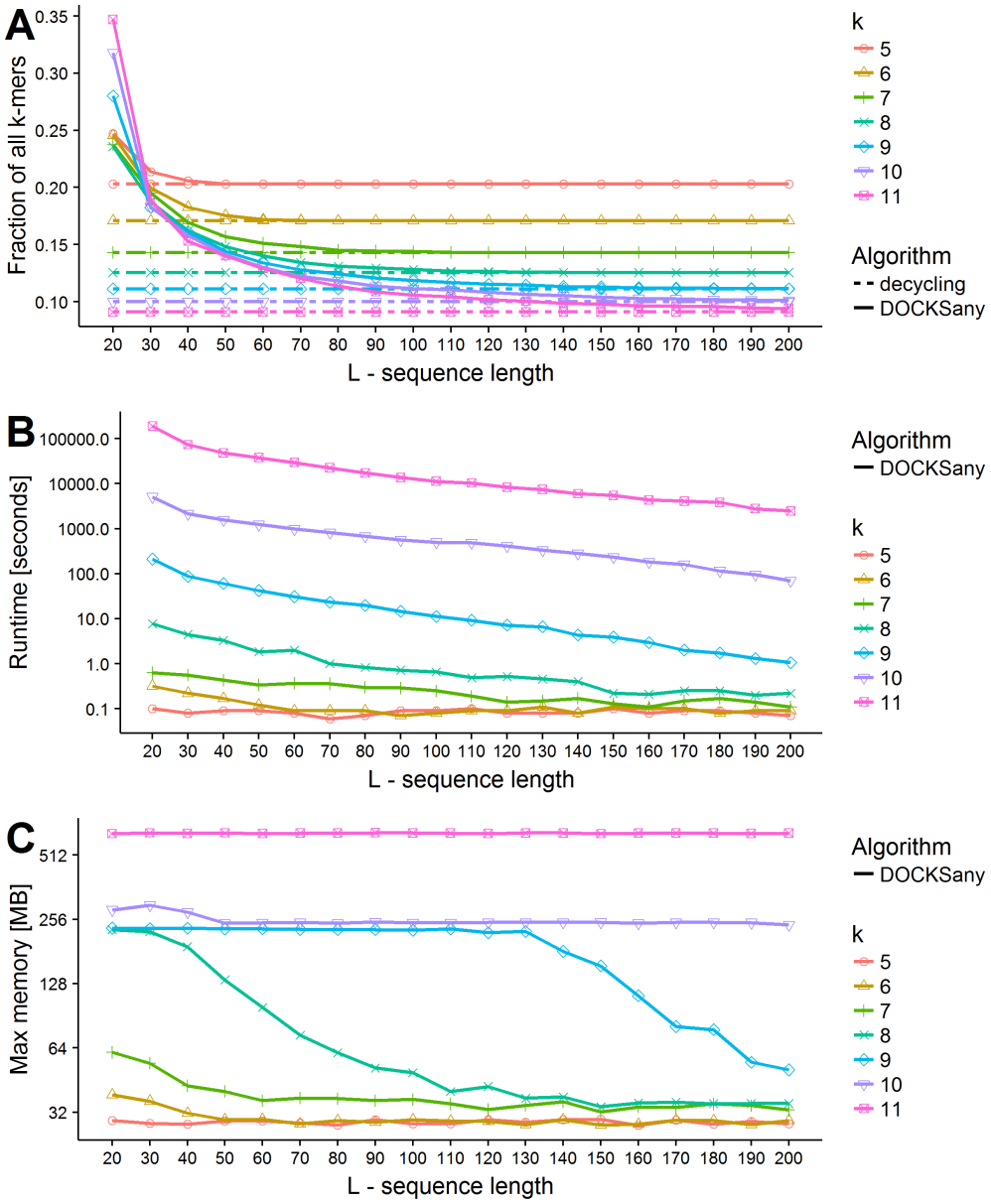
Supporting Information

Fig A



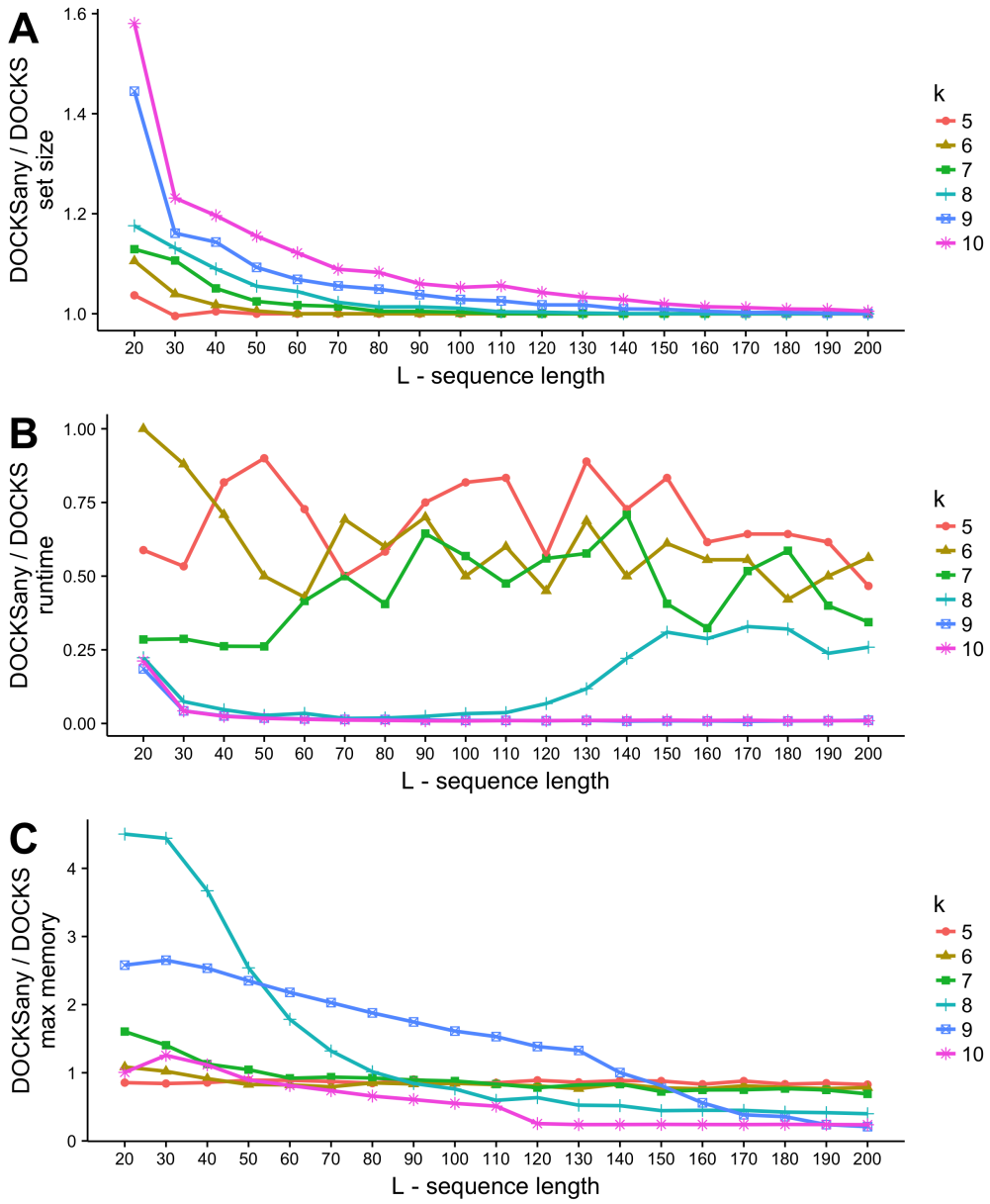
Performance of the greedy algorithm compared to DOCKS. The graphs show the ratio between the greedy algorithm and DOCKS in terms of (A) the k-mer set size generated; (B) the runtime used; (C) the max memory used.

Fig B



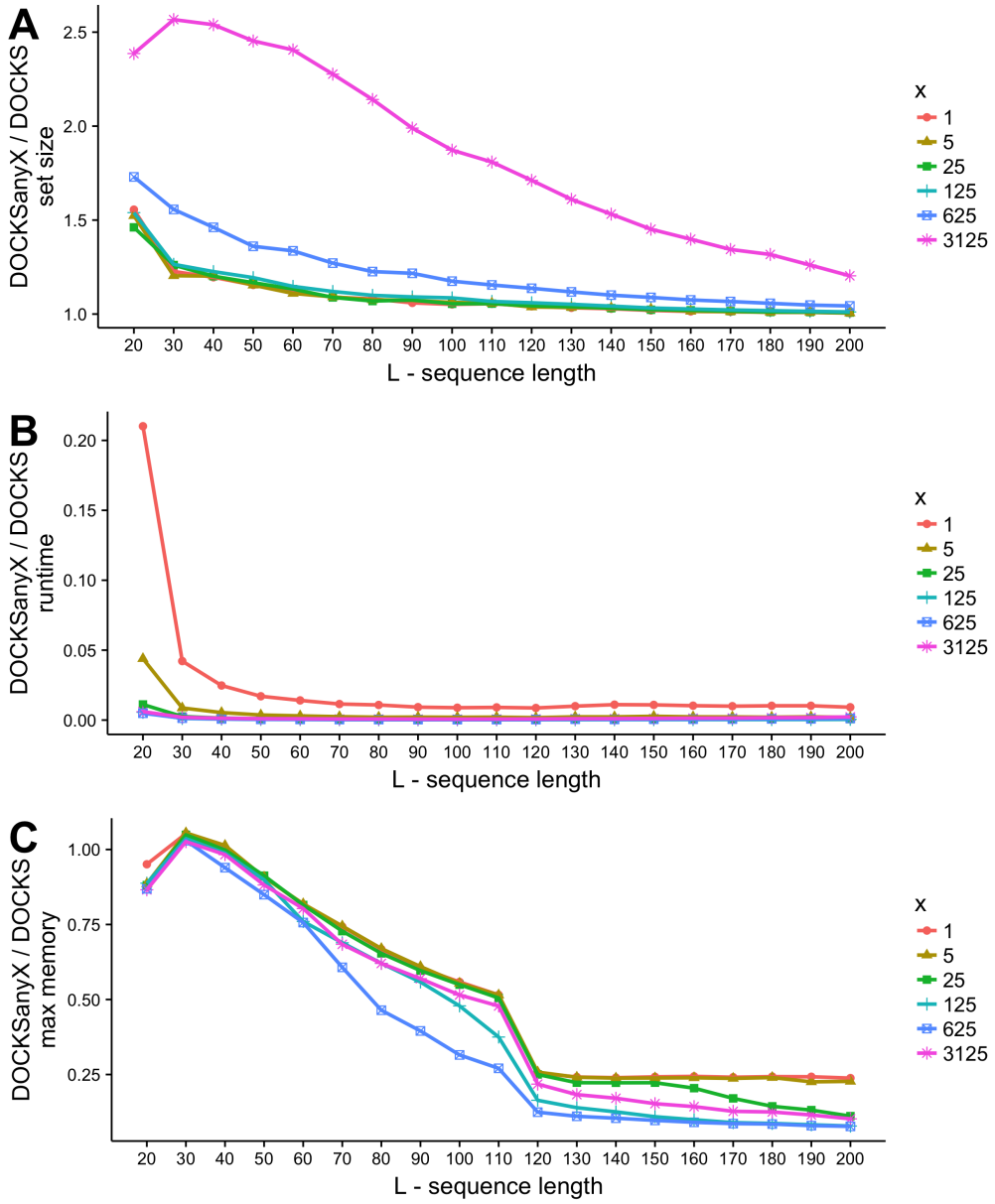
Performance of DOCKSany. For different combinations of k and L we ran DOCKSany over the DNA alphabet. (A) Set sizes. The results are shown as a fraction of the total number of k-mers $|\Sigma|^k$. The broken lines show the decycling set size for each k. (B) Running time in seconds. Note that y-axis is in log scale. (C) Maximum memory usage in megabytes. Note that y-axis is in log scale.

Fig C



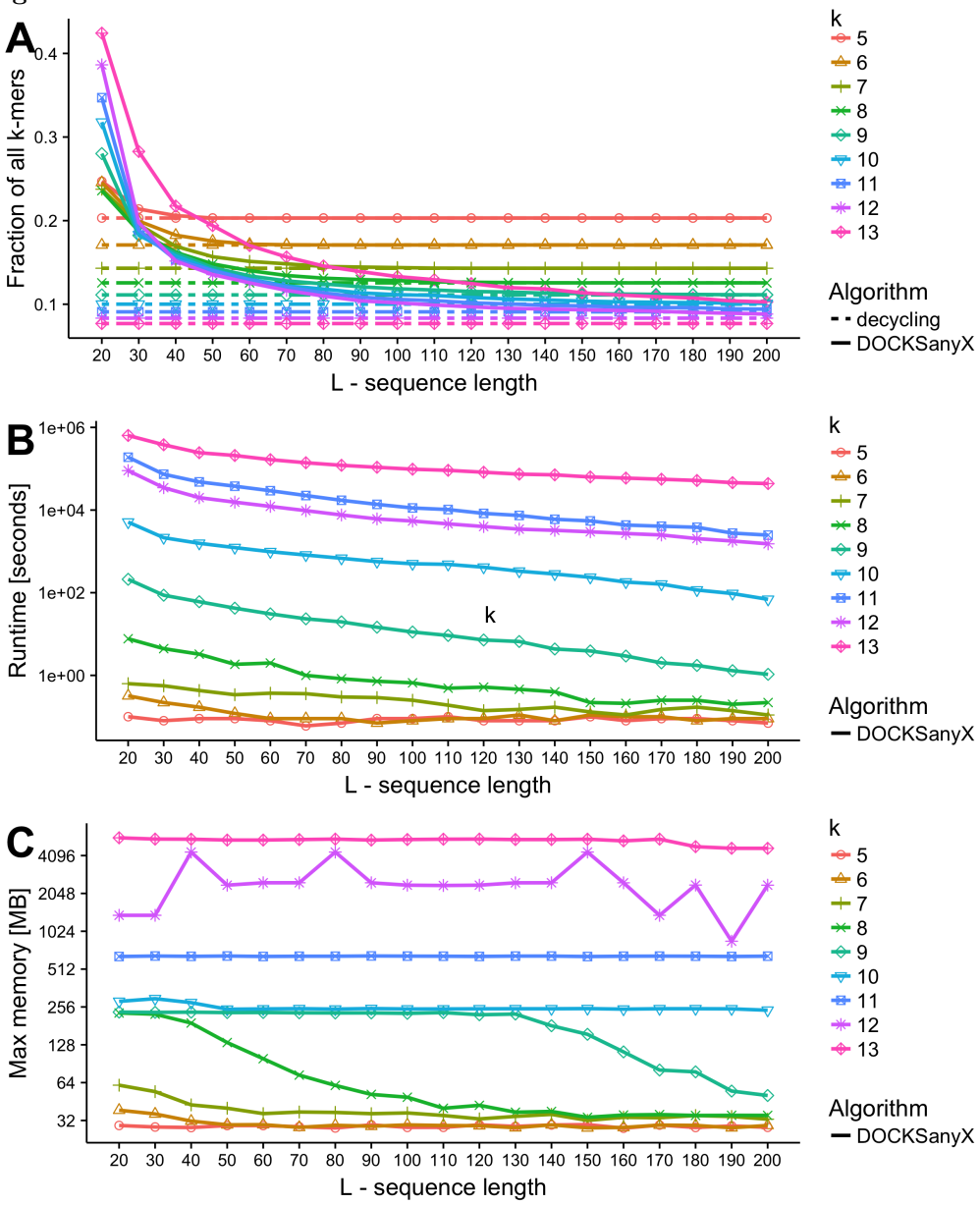
Performance of DOCKSany compared to DOCKS. The graphs show the ratio between DOCKSany and DOCKS for (A) the k-mer set size generated; (B) the runtime used; (C) the maximum memory used.

Fig D



Performance of DOCKSanyX. The graphs show the ratio between DOCKSanyX and DOCKS on $k = 10$ for (A) the k-mer set size generated; (B) the runtime used; (C) the maximum memory used.

Fig E



Performance of DOCKSanyX. For different combinations of k and L we ran DOCKSanyX over the DNA alphabet. X value was 1 for $5 \leq k \leq 11$, 100 for $k = 12$ and 10000 for $k = 13$. (A) Set sizes. The results are shown as a fraction of the total number of k-mers $|\Sigma|^k$. The broken lines show the decycling set size for each k. (B) Running time in seconds. Note that y-axis is in log scale. (C) Maximum memory usage in megabytes. Note that y-axis is in log scale.

Appendix

In this section we prove theoretical results used in the body of the paper.

NP-hardness of MINIMUM (k, L)-HITTING SET

One of the motivations for a universal k-mer set comes from the fact that the problem of finding a minimum-size k-mer set that hits every string in a given set of L-long strings is NP-hard. The hitting set problem, if a given set of target sequences is part of the input, is as follows:

MINIMUM (k, L)-HITTING SET

INSTANCE: Set S of L-long sequences over Σ and k.

VALID SOLUTION: Set X of k-mers s.t. $S \subseteq hit(X, L)$.

GOAL: Minimize $|X|$.

We prove that MINIMUM (k, L)-HITTING SET is NP-hard. For simplicity, we study the problem on the DNA alphabet, but it can be easily generalized to any finite alphabet Σ. We show a reduction from HITTING SET [1]. While the problems look similar, HITTING SET is a more general case than our problem, since in HITTING SET the subsets are arbitrary, while in MINIMUM (k, L)-HITTING SET problem each subset is made of overlapping k-mers. Hence, the hardness of the former does not directly imply hardness of the latter.

Theorem 1. *MINIMUM (k, L)-HITTING SET is NP-hard.*

Proof. Given an input to HITTING SET, a set S of subsets of $E = \{e_1 \dots e_n\}$, we generate an input to MINIMUM (k, L)-HITTING SET problem as follows: Denote by m the size of the maximum cardinality set, i.e. $m = \max_{S_i \in S} |S_i|$. We choose $\ell = \lceil \log_2(\max(m, n)) \rceil$, $L = 3\ell m$ and $k = 2\ell$. We map each set $S_i \in S$ to a k-long binary representation of i, where instead of bits we use nucleotides `C` and `G`. We map each element $e_j \in E$ to a k-long binary representation of j, where instead of bits we use nucleotides `A` and `T`. We call these representations the set's $\{C, G\}$-representation and the element's $\{A, T\}$-representation and denote them by $f_{CG}(S_i)$ and $f_{AT}(e_j)$.

We generate a sequence set T, which is the input to MINIMUM (k, L)-HITTING SET. For each set $S_i \in S$ we generate a sequence that contains all of its elements' $\{A, T\}$-representations, each appearing twice consecutively and buffered by the set's $\{C, G\}$-representation. Formally, for the set $S_i = \{e_{i_1}, \ldots, e_{i_{|S_i|}}\}$ we create the sequence: $T_i := (\prod_{j=1}^{|S_i|} f_{AT}(e_{i_j}) \cdot f_{AT}(e_{i_j}) \cdot f_{CG}(S_i)) \cdot (f_{AT}(e_{i_1}) \cdot f_{AT}(e_{i_1}) \cdot f_{CG}(S_i))^{m-|S_i|}$ (here $\prod$ indicates concatenation). The new instance T is $\{T_1, \ldots, T_{|S|}\}$.

Denote by T^{OPT} an optimal solution to MINIMUM (k, L)-HITTING SET. If a k-mer contains as a substring a complete $\{A, T\}$-representation w, then the element $f_{AT}^{-1}(w)$ is in the optimal solution to HITTING SET. If a k-mer contains a complete $\{C, G\}$-representation w, then any element from the set $f_{CG}^{-1}(w)$ can be part of the optimal solution. The running time of the reduction is bounded by $O(|S| \times L)$ to generate the input sequence set T. In terms of m and n the running time is $O(|S| \cdot m \cdot (\log(m) + \log(n)))$.

We now prove the correctness of the reduction. We start with proving several properties of the solution.

Lemma 1. *A k-mer that contains a complete $\{A, T\}$-representation w can be replaced by k-mer ww to produce a hitting set of the same cardinality.*

Proof. The k-mer contains a complete $\{A, T\}$-representation w. Thus, it can only hit sequences that contain w. Since the sequences were constructed to contain two adjacent $\{A, T\}$-representations per element, and since this representation is unique, k-mer ww hits the same set of sequences. □

Lemma 2. *A k-mer that contains a complete $\{C, G\}$-representation can be replaced by a k-mer that contains two adjacent occurrences of any $\{A, T\}$-representation from this sequence to produce a hitting set of the same cardinality.*

Proof. A $\{C, G\}$-representation is unique to each sequence. Thus, it can only hit one sequence, and replacing it by any other k-mer from that sequence preserves the hitting properties of the set. □

We now prove the two sides of the reduction:

1. MINIMUM (k, L)-HITTING SET ⇒ HITTING SET: all L-long sequences in T are hit by k-mers in T^{OPT}. By Lemmas 1 and 2 we can transform any hitting set

to a hitting set of the same cardinality, but containing only k-mers over $\{A, T\}$. These correspond to elements in an optimal solution of HITTING SET. Assume contrary that there is a smaller solution U to HITTING SET. Then, the set $\{f_{AT}(w) \cdot f_{AT}(w) \mid w \in U\}$ hits all sequences in the k-mer hitting problem, and by that producing a smaller solution, contrary to its optimality.

2. HITTING SET $\Rightarrow$ MINIMUM (k, L)-HITTING SET: denote by S^{OPT} an optimal solution to HITTING SET. Then, a set of k-mers $\{f_{AT}(w) \cdot f_{AT}(w) \mid w \in S^{OPT}\}$ is an optimal solution to MINIMUM (k, L)-HITTING SET. Assume contrary that there is a smaller solution U to MINIMUM (k, L)-HITTING SET. By Lemmas 1 and 2 there is a solution composed of k-mers over $\{A, T\}$. The set of element $\{f_{AT}^{-1}(w_{1:k/2}) \mid w \in U\}$ is a smaller hitting set in HITTING SET, contrary to its optimality.

□

NP-hardness of MINIMUM ℓ-PATH COVER IN A DAG

Our heuristic to find $U_{k,L}$ searches for a minimum ℓ-path cover in the DAG created after removing a decycling set. In the second phase of DOCKS we encounter a special case of the following problem.

MINIMUM ℓ-PATH VERTEX COVER IN A DAG

INSTANCE: A directed acyclic graph $G = (V, E)$ and integer ℓ.

VALID SOLUTION: Vertex set X s.t. $G' = (V \setminus X, E)$ contains no ℓ-long paths.

GOAL: Minimize $|X|$.

This general problem was shown to be NP-hard in [2]. A special case of the problem, for an acyclic subgraph of the de Bruijn graph, arises in the second phase of DOCKS after removing a minimum decycling set. The hardness result motivates the use of heuristics in the second phase.

Validity of the ILP formulation

Lemma 3. *The ILP is a valid formulation of the minimum hitting set problem.*

Proof. Suppose S is a UHS, and define $x^*_v = 1 \iff v \in S$, $L^*_v = 0$ if $v \in S$ and otherwise L^*_v equal to the length of the longest path ending at v. We claim that (x^*, L^*) satisfy the constraints. By construction, (8) holds. To show (9), if $v \in S$ then $0 = L^*_v \geq 1 + L^*_u - \ell$. If $v \notin S$, then $L^*_v \geq 1 + L^*_u$ by the property of the longest path labels. Hence all constraints are satisfied. Conversely, suppose the vectors x^* and L^* solve the ILP. W.l.o.g., we can assume that L^* is integer (otherwise round all coordinates down and all inequalities still hold for the new solution). Define $S = \{i \mid x^*_i = 1\}$. We claim that S is a UHS. Suppose by contradiction there exists a path of ℓ edges $p = (u_0, e_0, u_1, e_1, \ldots, u_\ell)$ in the graph induced by $G_k \setminus S$ (i.e. the DAG induced by removing the set S from the order k de Bruijn graph). Then, $x^*_{u_i} = 0$ for $i = 0, \ldots, \ell$ and summing the inequalities (9) for the edges in the path we get $L_{u_\ell} \geq L_{u_0} + \ell$, which contradicts (8). Hence, S is indeed a UHS. □

References

1. Karp RM. Reducibility among combinatorial problems. In: 50 Years of Integer Programming 1958-2008. Springer; 2010. p. 219–241.

2. Paindavoine M, Vialla B. Minimizing the Number of Bootstrappings in Fully Homomorphic Encryption. In: Revised Selected Papers of the 22Nd International Conference on Selected Areas in Cryptography - SAC 2015 - Volume 9566. New York, NY, USA: Springer-Verlag New York, Inc.; 2016. p. 25–43. Available from: http://dx.doi.org/10.1007/978-3-319-31301-6_2.

Chapter 3

PlasClass improves plasmid sequence classification

PLOS COMPUTATIONAL BIOLOGY

RESEARCH ARTICLE

PlasClass improves plasmid sequence classification

David Pellow[1]*, **Itzik Mizrahi**[2], **Ron Shamir**[1]*

1 Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel, **2** Department of Life Sciences, Ben-Gurion University of the Negev and the National Institute for Biotechnology in the Negev, Marcus Family Campus, Beer-Sheva, Israel

* dpellow@tau.ac.il (DP); rshamir@tau.ac.il (RS)

OPEN ACCESS

Citation: Pellow D, Mizrahi I, Shamir R (2020) PlasClass improves plasmid sequence classification. PLoS Comput Biol 16(4): e1007781. https://doi.org/10.1371/journal.pcbi.1007781

Editor: Mihaela Pertea, Johns Hopkins University, UNITED STATES

Received: December 24, 2019

Accepted: March 8, 2020

Published: April 3, 2020

Data Availability Statement: The reported results are for publicly available datasets. The waste-water plasmidome and human gut metagenomes are available through the SRA (accessions ERR1538272, ERR1297700, ERR1297720, ERR1297770, ERR1297796, ERR1297822, ERR1297834). The bacterial isolates dataset was previously curated from publicly available data by Arredondo-Alonso et al. It can be accessed from https://gitlab.com/sirarredondo/Plasmid_Assembly.

Funding: DP is supported in part by an Edmond J. Safra PhD Fellowship (https://safrabio.cs.tau.ac.il/),

Abstract

Many bacteria contain plasmids, but separating between contigs that originate on the plasmid and those that are part of the bacterial genome can be difficult. This is especially true in metagenomic assembly, which yields many contigs of unknown origin. Existing tools for classifying sequences of plasmid origin give less reliable results for shorter sequences, are trained using a fraction of the known plasmids, and can be difficult to use in practice. We present PlasClass, a new plasmid classifier. It uses a set of standard classifiers trained on the most current set of known plasmid sequences for different sequence lengths. We tested PlasClass sequence classification on held-out data and simulations, as well as publicly available bacterial isolates and plasmidome samples and plasmids assembled from metagenomic samples. PlasClass outperforms the state-of-the-art plasmid classification tool on shorter sequences, which constitute the majority of assembly contigs, allowing it to achieve higher F1 scores in classifying sequences from a wide range of datasets. PlasClass also uses significantly less time and memory. PlasClass can be used to easily classify plasmid and bacterial genome sequences in metagenomic or isolate assemblies. It is available under the MIT license from: https://github.com/Shamir-Lab/PlasClass.

This is a *PLOS Computational Biology* Software paper.

Introduction

When using high-throughput sequencing to study the presence and dynamics of plasmids in their bacterial hosts, it is often necessary to classify sequences as being of plasmid or chromosomal origin. This is especially true in the case of metagenomic sequencing, which can include many sequences of unknown origin and varying lengths. We focus on the challenge of classifying contigs in a metagenomic assembly in order to identify which are of plasmid origin.

The current state-of-the-art classifier of plasmid sequences is PlasFlow [1], a neural network based algorithm that was shown to perform better than previous tools such as cBar [2]. While PlasFlow is successful in classifying small sets of long sequences, it produces less reliable results for short sequences and requires large memory on very large metagenomic datasets.

and in part by an Israel Ministry of Immigrant Absorption PhD fellowship (https://www.gov.il/en/departments/general/research_students_scholarship). RS is supported in part by grants from the Israel Science Foundation (ISF - https://www.isf.org.il/#/) grant 1339/18, the US - Israel Binational Science Foundation (BSF - https://www.bsf.org.il/), and the US National Science Foundation (NSF - https://www.nsf.gov/) grant 2016694. IM is supported in part by ISF grant 1947/19 (ISF - https://www.isf.org.il/#/) and ERC Horizon 2020 research and innovation program grant 640384 (https://ec.europa.eu/programmes/horizon2020/en). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Here we present PlasClass, a new plasmid sequence classifier implemented as an easy to use Python package. It uses a set of logistic regression classifiers each trained on sequences of a different length sampled from plasmid and bacterial genome reference sequences. When applied on a set of sequences, the appropriate length-specific classifier is used for each sequence.

We tested PlasClass on simulated data, on bacterial isolates, on a wastewater plasmidome, and on plasmids assembled from human gut microbiome samples. For shorter sequences, which are the majority of contigs in an assembly, PlasClass achieved better F1 scores than PlasFlow. This resulted in better overall performance across all the datasets tested. PlasClass also used significantly less RAM and disk memory than PlasFlow, and can be run much faster by using multiprocessing.

PlasClass is provided at https://github.com/Shamir-Lab/PlasClass.

Design and implementation

Training databases

We used reference sequence databases to obtain the training sequences for our classifiers. For the plasmid references we used plasmid sequences listed in PLSDB [3] (v.2018_12_05), an up-to-date curated plasmid database. After filtering out duplicate sequences this database contained 13469 reference plasmids (median length: 53.8kb).

For the bacterial chromosome references we downloaded all complete bacterial genome assemblies from NCBI (download date January 9, 2019). We removed sequences annotated as being plasmids and filtered out duplicates, leaving 13491 reference chromosomes (median length: 3.7Mbp).

One quarter of the sequences were randomly removed from the databases before training in order to provide a held-out test set for validation. PlasClass was retrained on the full databases and this version was used for testing on assembled data.

Training the classifiers

We sampled sequence fragments of different lengths from the reference sequences with replacement and constructed a k-mer frequency vector for each fragment. Canonical k-mers of lengths 3–7 were used, resulting in a feature vector of length 10952 for each fragment. Fragment lengths were 500k, 100k, 10k, and 1k. For the two shorter lengths, 90,000 training fragments were used from each class. For the lengths 500k and 100k, since there were not enough long plasmids to do the same, we sampled enough fragments to cover all of the sufficiently long plasmids to a depth of 5. This resulted in 1934 and 45525 plasmid fragments of length 500k and 100k, respectively on the full plasmid database.

For each length, a logistic regression classifier was trained on the plasmid and chromosomal fragments' k-mer frequency vectors using the scikit-learn [4] machine learning library in Python. Code is provided to retrain the models on user-supplied reference sequence databases.

Length-specific classification

PlasClass uses four logistic regression models to classify sequences of different length. Each sequence is assigned to the closest length from among 1kb, 10kb, 100kb, and 500kb. Equivalently, this defines four length ranges: (0,5.5kb], (5.5kb,55kb], (55kb,300kb], (300kb, ∞). Given a sequence, its k-mers are counted, the canonical k-mer frequency vector is calculated and used to classify it with the classifier for the range it falls into. k-mer counting can be performed in parallel for different sequences. Finally, all classification results are concatenated into a single output in the same order as the input sequences.

Classification with PlasClass

PlasClass is available at https://github.com/Shamir-Lab/PlasClass. It has been retrained using the full set of database references. PlasClass can be used as a command-line tool to classify sequences in an input fasta file or it can be imported as a module into the user's code to classify sequences in the user's program. It can be run in parallel mode to achieve faster runtimes. PlasClass is fully documented in S1 File and at the url provided above.

Results

We tested performance of PlasClass on both simulated and real data and compared it to PlasFlow.

Experimental settings

PlasClass and PlasFlow both assign class probabilities to each sequence. We say a sequence is classified as having plasmid origin if the probability that it belongs to the plasmid class is > 0.5. When running PlasFlow, this probability was summed over all plasmid classes, and we set the parameter `--threshold = 0.5` to ensure each sequence is classified as either plasmid or bacterial. All assemblies were performed using the `--meta` option of SPAdes [5] v3.12.

Performance metrics

We calculated the precision, recall and F1 scores counting the *number* of true positive and false positive predictions. Some previous works [1, 6] calculated performance based on the *lengths* of the sequences classified as plasmids and the total length of the plasmids in a sample. A length-weighted metric is appropriate in the context of plasmid sequence assembly, but in the context of contig classification this makes little sense. (Consider the extreme case of one extremely long sequence and 999 very short ones. Classifying the long contig is easy, but a classifier that only identifies it correctly will have weighted precision and recall near 1 even though only 1/1000 of the sequences are correctly classified.) For this reason we used the numbers of correctly classified sequences.

On the assembled contigs we follow the previous works [1, 6] and consider a contig to be from the plasmid class if it matches a plasmid reference sequence—even if it also matches a chromosomal reference sequence. This is appropriate for classifying all sequences in an assembly to determine their origin. However, when constructing a benchmark for a classifier, it may be more suitable to filter ambiguous sequences that may belong to both classes out of the test set. For this reason, we also report results with all ambiguous sequences filtered out in S2 File.

Classifying sequences from held-out references

We sampled overlapping L-long fragments covering the held out plasmids with an overlap of $L/2$ for L = 100k, 10k and 1k. A matching number of L-long fragments were sampled from the held out bacterial genomes for each length L. (Note that this creates a balanced classification scenario.) Table 1 summarizes the classification results. PlasClass improved precision at the cost of slightly lower recall and had better overall F1 on the shorter sequence lengths. These short sequences can make up the majority of contigs in metagenomic assemblies, allowing PlasClass to outperform PlasFlow in many settings as shown below.

Table 1. Performance on held out data.

Length (bp)	# fragments per class	PlasClass			PlasFlow		
		Precision	Recall	F1	Precision	Recall	F1
100k	2979	96.9	85.4	90.8	95.6	88.4	91.9
10k	56583	88.7	86.4	87.6	83.1	87.7	85.3
1k	607656	75.1	74.6	74.8	59.7	79.1	68.1

Performance of PlasClass and PlasFlow on fixed length sequence fragments sampled from the held out references.

https://doi.org/10.1371/journal.pcbi.1007781.t001

Performance on a benchmark of bacterial isolates

We compared the performance of PlasClass to PlasFlow on the isolate assemblies from the benchmark in [6]. Specifically, we downloaded the assemblies and all bacterial and plasmid reference sequences used in the benchmarking experiment of [6] (available from: https://gitlab.com/sirarredondo/Plasmid_Assembly). Assembled contigs were mapped to the references using BLAST and contigs with matches (>95% mapping identity along >95% of the contig length) were assigned to the plasmid or chromosome class as described. There were 60579 contigs across all the assemblies of which 36172 matched one of the classes (8569 plasmid and 27603 chromosome) and were used in this test. As seen in Table 2, the majority of these sequences were extremely short (68% of the 36172 contigs <500bp). We looked at the impact of these short sequences by filtering out contigs below a certain length and the results of both methods improved when shorter sequences were filtered out. In all cases, PlasClass had consistently higher F1.

Performance on simulated metagenome assemblies

We simulated metagenomes by randomly selecting bacterial genome references from the NCBI along with their associated plasmids and using realistic distributions for genome abundance and plasmid copy number. For genome abundance we used the log-normal distribution, normalized so that the relative abundances sum to 1. For plasmid copy number we used a geometric distribution with parameter $p = min(1, log(L)/7)$ where L is the plasmid length. This makes it much less likely for a long plasmid to have a copy number above 1, while shorter plasmids can have higher copy numbers. Short reads were simulated from the genome references using InSilicoSeq [7] and assembled.

We then classified the assembled contigs. Classification was performed on the assembled contigs that had a match to either a reference plasmid or reference chromosome sequence used in the simulation (1641 plasmid contigs, 32451 chromosome contigs in Sim1, and 14272

Table 2. Performance on bacterial isolates.

Contig length (bp)	# of contigs	PlasClass			PlasFlow		
		Precision	Recall	F1	Precision	Recall	F1
All	36172	43.65	77.58	55.87	31.16	87.77	46.00
>500	11659	53.15	91.30	67.18	37.68	89.23	52.99
>1000	7414	59.95	91.82	72.54	47.54	90.04	62.23
>5000	3999	61.84	92.12	74.00	50.05	92.31	64.91

Performance on bacterial isolates from [6], as a function of the minimum contig length.

https://doi.org/10.1371/journal.pcbi.1007781.t002

Table 3. Performance on simulated metagenomes.

	# chromosomes	# plasmids	# unique	# contigs	PlasClass F1	PlasFlow F1
Sim1	34	82	56	34092	15.79	13.49
Sim2	198	333	219	388669	12.08	8.79

Summary of the simulated metagenome datasets and comparison of F1 scores. # unique is the number of distinct plasmids, ignoring multiple copies.

https://doi.org/10.1371/journal.pcbi.1007781.t003

plasmid contigs, 374397 chromosome contigs in Sim2). F1 results are shown in Table 3. PlasClass outperformed PlasFlow by more than 17%. Scores were low for both methods due to the many short contigs in the assembly (50% and 73% of the contigs <500 bp in Sim1 and Sim2 respectively) and the class imbalance. We show the impact of short sequences on performance in Table 4. PlasClass consistently outperformed PlasFlow, and both methods performed better as more short sequences were filtered out.

Performance on a plasmidome sample

We assembled the wastewater plasmidome sample ERR1538272 from the study by Shi et al. [8]. It is a metagenomic sample that was enriched for plasmid sequences. Each contig in the assembly was matched to the plasmid and bacterial reference databases using BLAST. The set of 9854 contigs (out of 35285) that matched the reference sequences (1888 plasmid contigs, 7966 chromosome contigs) was used as the gold standard to test the classifiers (contig length distribution is presented in S3 File See also S1 Fig). Although the plasmid-enriched setting favors PlasFlow, which sacrifices precision for higher recall, PlasClass still had a higher combined F1 as shown in Table 5.

We computed the precision-recall curve for the classification of the gold standard contigs in this sample by PlasClass, shown in S2 Fig (see also S3 File. The area under the curve is 0.41, more than double the baseline of 0.19 (the fraction of the contigs that are of plasmid origin).

Classifying plasmids assembled from metagenomic samples

We assembled six publicly available human gut microbiome samples (accessions: ERR1297700, ERR1297720, ERR1297770, ERR1297796, ERR1297822, ERR1297834) and found plasmid sequences in the assemblies using Recycler [9]. Recycler assembles plasmid sequences based on coverage and circularity—features that are not used by the classifiers. 16–

Table 4. Simulated metagenome performance by length.

	Contig length (bp)	# of contigs	PlasClass			PlasFlow		
			Precision	Recall	F1	Precision	Recall	F1
Sim1	All	34092	8.94	67.40	15.79	7.30	87.75	13.49
	>500	17023	11.22	78.55	19.64	8.20	85.05	14.95
	>1000	11696	15.67	80.96	26.26	10.92	85.00	19.36
	>5000	4032	36.11	86.80	51.00	28.09	90.80	42.91
Sim2	All	388669	6.64	66.98	12.08	4.64	84.31	8.79
	>500	106814	13.76	76.00	23.29	8.42	84.23	15.32
	>1000	45597	22.42	79.20	34.95	14.01	86.52	24.11
	>5000	5642	46.50	81.18	59.13	38.48	88.49	53.63

Performance on simulated metagenomes as a function of the minimum contig length.

https://doi.org/10.1371/journal.pcbi.1007781.t004

Table 5. Performance on a plasmidome sample.

	Precision	Recall	F1 score
PlasClass	**32.32**	64.25	**43.01**
PlasFlow	23.72	**86.49**	37.23

Performance of PlasClass and PlasFlow on the plasmidome sample from [8].

https://doi.org/10.1371/journal.pcbi.1007781.t005

27 plasmids were assembled per sample (median length: 3.4kb). We classified each of the plasmids generated by Recycler to determine the extent of agreement between the sequence classifiers and this orthogonal approach. As seen in Fig 1, PlasClass agreed with Recycler on the same number or more plasmids than PlasFlow in all samples. This suggests that PlasClass can correctly identify more plasmids in real datasets, which contain many previously unknown plasmid sequences.

Resource usage

In Table 6, we compare the runtime and memory usage of PlasClass and PlasFlow on the full plasmidome, simulated metagenome, and isolate bacterial datasets. PlasClass (running with a

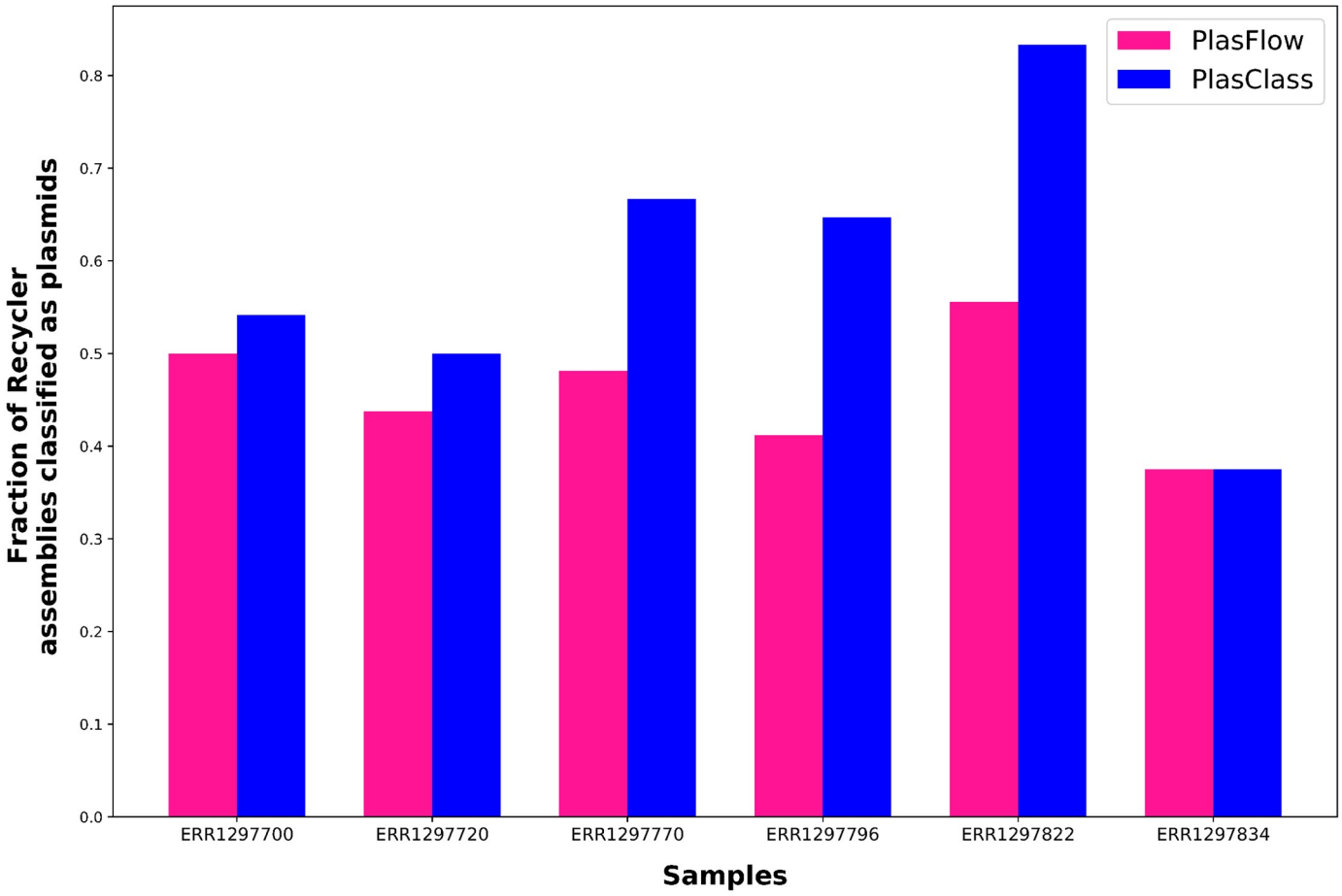


Fig 1. Classifying plasmids assembled from metagenomic samples. Agreement of PlasClass and PlasFlow classifications with the plasmids generated by Recycler.

https://doi.org/10.1371/journal.pcbi.1007781.g001

Table 6. Resource usage.

Dataset	PlasFlow			PlasClass		PlasClass—8 processes	
	Runtime	RAM	Disk	Runtime	RAM	Runtime	RAM
Isolates	12.8	47.8	21.4	36.3	17.2	6.8	17.2
Sim1	7.1	28.3	12.1	16.2	12.0	3.0	12.0
Sim2	89.3	291.3	137.5	54.8	17.3	17.1	17.3
Plasmidome	7.9	28.8	12.2	4.2	12.2	5.2	17.3

Runtime (wall clock time, in minutes) and memory usage (in GB) of PlasClass and PlasFlow.

https://doi.org/10.1371/journal.pcbi.1007781.t006

single process) was faster than PlasFlow on the most time consuming sample and was significantly faster in all cases when using multiprocessing. It used less than half the RAM of PlasFlow and the RAM usage was not increased significantly when using multiprocessing. PlasFlow writes the feature matrices to disk while PlasClass does not. Performance was measured on a 44-core, 2.2 GHz server with 792 GB of RAM.

Discussion

We presented the PlasClass algorithm for classifying plasmid sequences. We applied the algorithm across a wide range of contexts and showed that in most cases PlasClass outperformed the state-of the-art algorithm PlasFlow. It was also faster and required less memory.

The task of classifying plasmid sequences in the real-world context of metagenomic data is a difficult task due to the nature of the assembled sequences: the sequences are mostly short (60-90% are shorter than 1 kbp, see Tables 2 and 4), and there is an imbalance between the number of plasmid and bacterial sequences (1:3 in the bacterial isolates, and 1:4 in the plasmid-enriched plasmidome samples presented). Given the constraints, the quality of classification is naturally limited, but the task is of high importance for understanding plasmid role in horizontal transfer, antibiotic resistance and ecology. We also showed that classification quality improves when focusing on longer sequences and when plasmid sequences are enriched.

Availability and future directions

PlasClass is open-source and freely available under the MIT license. PlasClass is maintained on GitHub, enabling bug-reporting and community collaboration in extending the tool to meet needs of the users as they arise. It can be found at https://github.com/Shamir-Lab/PlasClass.

We plan to use PlasClass in order to improve plasmid assembly from metagenomic samples, by utilizing the classification scores of contigs. Another possible future direction is to tailor the plasmid training data to the problem at hand: Currently we use all known plasmids for training, which creates a bias towards clinically relevant samples. By using training datasets tailored to other specific environments one can create a classifier that would fit those environments better.

Supporting information

S1 File. PlasClass documentation. Complete documentation for using PlasClass.
(PDF)

S2 File. Results with ambiguous sequences filtered. Extended results reporting performance with ambiguous sequences filtered out.
(PDF)

S3 File. Plasmidome dataset extended results. Extended results reporting the contig lengths and precision-recall curve for the plasmidome sample.
(PDF)

S1 Fig. Plasmidome contig lengths. Histogram of the contig lengths in the plasmidome assembly. Note that the y-axis uses log-scale.
(TIF)

S2 Fig. Plasmidome precision-recall curve. Precision-recall curve for the classification of contigs of in the plasmidome sample.
(TIF)

Acknowledgments

We thank the members of the Shamir and Mizrahi Labs for their help and advice.

Author Contributions

Conceptualization: David Pellow, Itzik Mizrahi, Ron Shamir.

Data curation: David Pellow.

Formal analysis: David Pellow.

Funding acquisition: Itzik Mizrahi, Ron Shamir.

Methodology: David Pellow.

Project administration: Ron Shamir.

Software: David Pellow.

Supervision: Ron Shamir.

Validation: David Pellow.

Writing – original draft: David Pellow, Ron Shamir.

Writing – review & editing: David Pellow, Ron Shamir.

References

1. Krawczyk PS, Lipinski L, Dziembowski A. PlasFlow: predicting plasmid sequences in metagenomic data using genome signatures. Nucleic Acids Research. 2018; 46(6):e35. https://doi.org/10.1093/nar/gkx1321 PMID: 29346586
2. Zhou F, Xu Y. cBar: a computer program to distinguish plasmid-derived from chromosome-derived sequence fragments in metagenomics data. Bioinformatics. 2010; 26(16):2051–2052. https://doi.org/10.1093/bioinformatics/btq299 PMID: 20538725
3. Galata V, Fehlmann T, Backes C, Keller A. PLSDB: a resource of complete bacterial plasmids. Nucleic Acids Research. 2018; 47(D1):D195–D202.
4. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011; 12:2825–2830.
5. Bankevich A, Nurk S, Antipov D, Gurevich AA, Dvorkin M, Kulikov AS, et al. SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. Journal of Computational Biology. 2012; 19(5):455–477. https://doi.org/10.1089/cmb.2012.0021 PMID: 22506599
6. Arredondo-Alonso S, Willems RJ, van Schaik W, Schürch AC. On the (im)possibility of reconstructing plasmids from whole-genome short-read sequencing data. Microbial genomics. 2017; 3(10):e000128. https://doi.org/10.1099/mgen.0.000128 PMID: 29177087
7. Gourlé H, Karlsson-Lindsjö O, Hayer J, Bongcam-Rudloff E. Simulating Illumina metagenomic data with InSilicoSeq. Bioinformatics. 2018; 35(3):521–522.

8. Shi Y, Zhang H, Tian Z, Yang M, Zhang Y. Characteristics of ARG-carrying plasmidome in the cultivable microbial community from wastewater treatment system under high oxytetracycline concentration. Applied microbiology and biotechnology. 2018; 102(4):1847–1858. https://doi.org/10.1007/s00253-018-8738-6 PMID: 29332216

9. Rozov R, Brown Kav A, Bogumil D, Shterzer N, Halperin E, Mizrahi I, et al. Recycler: an algorithm for detecting plasmids from de novo assembly graphs. Bioinformatics. 2017; 33(4):475–482. https://doi.org/10.1093/bioinformatics/btw651 PMID: 28003256

Chapter 4

SCAPP: an algorithm for improved plasmid assembly in metagenomes

Pellow *et al. Microbiome* (2021) 9:144
https://doi.org/10.1186/s40168-021-01068-z

Microbiome

SOFTWARE ARTICLE Open Access

SCAPP: an algorithm for improved plasmid assembly in metagenomes

David Pellow[1*], Alvah Zorea[2], Maraike Probst[3], Ori Furman[2], Arik Segal[4,5], Itzhak Mizrahi[2] and Ron Shamir[1]

Abstract

Background: Metagenomic sequencing has led to the identification and assembly of many new bacterial genome sequences. These bacteria often contain plasmids: usually small, circular double-stranded DNA molecules that may transfer across bacterial species and confer antibiotic resistance. These plasmids are generally less studied and understood than their bacterial hosts. Part of the reason for this is insufficient computational tools enabling the analysis of plasmids in metagenomic samples.

Results: We developed SCAPP (Sequence Contents-Aware Plasmid Peeler)—an algorithm and tool to assemble plasmid sequences from metagenomic sequencing. SCAPP builds on some key ideas from the Recycler algorithm while improving plasmid assemblies by integrating biological knowledge about plasmids.
We compared the performance of SCAPP to Recycler and metaplasmidSPAdes on simulated metagenomes, real human gut microbiome samples, and a human gut plasmidome dataset that we generated. We also created plasmidome and metagenome data from the same cow rumen sample and used the parallel sequencing data to create a novel assessment procedure. Overall, SCAPP outperformed Recycler and metaplasmidSPAdes across this wide range of datasets.

Conclusions: SCAPP is an easy to use Python package that enables the assembly of full plasmid sequences from metagenomic samples. It outperformed existing metagenomic plasmid assemblers in most cases and assembled novel and clinically relevant plasmids in samples we generated such as a human gut plasmidome. SCAPP is open-source software available from: https://github.com/Shamir-Lab/SCAPP.

Keywords: Plasmids, Assembly

Background

Plasmids play a critical role in microbial adaptation, such as antibiotic resistance or other metabolic capabilities, and genome diversification through horizontal gene transfer. However, plasmid evolution and ecology across different microbial environments and populations are poorly characterized and understood. Thousands of plasmids have been sequenced and assembled directly from isolated bacteria, but constructing complete plasmid sequences from short read data remains a hard challenge. The task of assembling plasmid sequences from shotgun metagenomic sequences, which is our goal here, is even more daunting.

There are several reasons for the difficulty of plasmid assembly. First, plasmids represent a very small fraction of the sample's DNA and thus may not be fully covered by the read data in high-throughput sequencing experiments. Second, they often share sequences with the bacterial genomes and with other plasmids, resulting in tangled assembly graphs. For these reasons, plasmids assembled from bacterial isolates are usually incomplete, fragmented into multiple contigs, and contaminated with sequences from other sources. The challenge is reflected in the title of a recent review on the topic: "On the (im)possibility of

*Correspondence: dpellow@post.tau.ac.il
[1]Blavatnik School of Computer Science, Tel Aviv University, 6997801 Tel Aviv, Israel
Full list of author information is available at the end of the article

reconstructing plasmids from whole-genome short-read sequencing data" [1]. In a metagenomic sample, these problems are amplified since the assembly graphs are much larger, more tangled, and fragmented.

There are a number of tools that can be used to detect plasmid sequences including PlasmidFinder [2], cBar [3], gPlas [4], PlasFlow [5], and others. There is also the plasmidSPAdes assembler for assembling plasmids in isolate samples [6]. However, there are currently only two tools that attempt to reconstruct complete plasmid sequences in metagenomic samples: Recycler [7] and metaplasmidSPAdes [8] (mpSpades). mpSpades iteratively generates smaller and smaller subgraphs of the assembly graph by removing contigs with coverage below a threshold that increases in each iteration. As lower coverage segments of the graph are removed, longer contigs may be constructed in the remaining subgraph. Cyclic contigs are considered as putative plasmids and then verified using the profile of their genetic contents. The main idea behind Recycler is that a single shortest circular path through each node in the assembly graph can be found efficiently. The circular paths that have uniform read coverage are iteratively "peeled" off the graph and reported as possible plasmids. The peeling process reduces the residual coverage of each involved node, or removes it altogether. We note that these tools, as well as our work, focus on circular plasmids and do not assemble linear plasmid sequences.

Here we present SCAPP (Sequence Contents-Aware Plasmid Peeler), a new algorithm that uses the peeling idea of Recycler and also leverages external biological knowledge about plasmid sequences. In SCAPP, the assembly graph is annotated with plasmid-specific genes (PSGs) and nodes are assigned weights reflecting the chance that they are plasmidic based on a plasmid sequence classifier [9]. In the annotated assembly graph, we prioritize peeling off circular paths that include plasmid genes and highly probable plasmid sequences. SCAPP also uses the PSGs and plasmid scores to filter out likely false positives from the set of potential plasmids.

Algorithm 1 SCAPP pipeline

Input: Assembly graph $G = (V, E)$ and read set R of the sample
Output: P: potential plasmids, O: confident plasmid predictions

Create annotated graph $G' = (V', E')$:
Initially $G' = G$
Map R to V'
$score(v) \leftarrow$ sequence plasmid probability $\forall v \in V'$
$w(v) = (1 - score(v))/(len(v) \cdot cov(v))\ \forall v \in V'$
$V^m = \{v \in V' | v$ contains a PSG$\}$, $w(v) = 0\ \forall v \in V^m$
$V' \leftarrow V' \setminus \{v \in V' |\, deg(v) = 0 \vee v$ is probable chromosome node
$\vee\ v$ is a non-compatible self-loop with $indeg(v) = outdeg(v) = 1\}$
$P \leftarrow \{v \in V' | v$ is a compatible self-loop$\}$
for each strongly connected component $CC \in G'$ **do**
    for $v \in V^m \cap CC$ in decreasing order by $len(v) \cdot cov(v)$ **do**
        Find lowest weight cycle C through v
        if C meets coverage and paired-end read criteria **then**
            $P \leftarrow P \cup \{C\}$, $G' \leftarrow peel(G', C)$
    for $v \in \{v \in CC |\, v$ is a probable plasmid node$\}$ in decreasing order by $len(v) \cdot cov(v)$ **do**
        Find lowest weight cycle C through v
        if C meets coverage and paired-end read criteria **then**
            $P \leftarrow P \cup \{C\}$, $G' \leftarrow peel(G', C)$
    while V' changes **do**
        $S \leftarrow \{\}$
        for $v \in V' \cap CC$ in decreasing order by $len(v) \cdot cov(v)$ **do**
            Find lowest weight cycle C through v
            $S \leftarrow S \cup C$
        for $C \in S$ in increasing order of coefficient of variation **do**
            if C meets coverage and paired-end read criteria **then**
                $P \leftarrow P \cup \{C\}$, $G' \leftarrow peel(G', C)$
$O \leftarrow \{C \in P | (C$ contains a PSG $\wedge$ *plasmid score*$(C) > 0.5)$
$\vee\ (C$ contains a PSG $\wedge\ C$ is self-loop $) \vee ($*plasmid score*$(C) > 0.5 \wedge C$ is self-loop$)\}$

We tested SCAPP on both simulated and diverse real metagenomic data and compared its performance to Recycler and mpSpades. Overall, SCAPP performed better than the other tools across these datasets. SCAPP has higher precision than Recycler in all cases, meaning it more accurately constructs correct plasmids from the sequencing data. SCAPP also has higher recall than mpSpades in most cases, and higher precision in most of the real datasets. We developed and tested a novel strategy given parallel plasmidome and metagenome sequencing of the same sample. We show how to accurately assess the performance of the tools on metagenome data, even in the absence of known reference plasmids.

Implementation

SCAPP accepts as input a metagenomic assembly graph, with nodes representing the sequences of assembled contigs and edges representing k-long sequence overlaps between contigs, and the paired-end reads from which the graph was assembled. SCAPP processes each component of the assembly graph and iteratively assembles plasmids from them. The output of SCAPP is a set of cyclic sequences representing confident plasmid assemblies.

A high-level overview of SCAPP is provided in Table 1 and depicted graphically in Fig. 1; the full algorithmic details are presented below. For brevity, we describe only default parameters below; see Additional file 1, Section S1 for alternatives.

SCAPP is available from https://github.com/Shamir-Lab/SCAPP and fully documented there. It was written in Python3 and can be installed as a conda package, directly from Bioconda or from its sources.

The SCAPP algorithm

The full SCAPP algorithm is given in Algorithm 1. The peel function, which defines how cycles are peeled from the graph, is given in Algorithm 2.

Algorithm 2 $peel(G, C)$

Input: Assembly graph $G = (V, E)$ annotated with node coverage, cycle $C \subset G$

Output: Updated graph $G' = (V' \subseteq V, E' \subseteq E)$ with cycle C peeled

1: $G' = G$
2: $\mu_{cov'}(C) = \sum_{u \in C} f(u, C) cov'(u, C)$, the weighted mean of the discounted coverage of C in G
3: **for** $v \in C$ **do**
4: $\quad cov(v) \leftarrow max\{cov(v) - \mu_{cov'}(C), 0\}$
5: $\quad$ **if** $cov(v) = 0$ **then**
6: $\quad\quad V' \leftarrow V' \setminus v$
7: $\quad\quad E' \leftarrow E' \setminus \{e | e = (u, v) \cup e = (v, u)\ \forall u \in V\}$

Table 1

Overview of SCAPP
1: Annotate the assembly graph:
a: Map reads to nodes of the assembly graph
b: Find nodes with plasmid-specific gene matches
c: Compute plasmid sequence scores of nodes
d: Assign node weights
2: **for** each strongly connected component **do**
3: Iteratively peel uniform coverage cycles through plasmid gene nodes
4: Iteratively peel uniform coverage cycles through high scoring nodes
5: Iteratively peel shortest cycle through each remaining node if it meets plasmid criteria
6: Output the set of confident plasmid predictions

Read mapping

The first step in creating the annotated assembly graph (Table 1 step 1a) is to align the reads to the contigs in the graph. The links between paired-end reads aligning across contig junctions are used to evaluate potential plasmid paths in the graph. SCAPP performs read alignment using BWA [10] and the alignments are filtered to retain only primary read mappings, sorted, and indexed using SAMtools [11].

Plasmid-specific gene annotation

We created sets of PSGs by database mining and curation by plasmid microbiology experts from the Mizrahi Lab (Ben-Gurion University). Information about these PSG sets is found in Additional file 1, Section S2. The sequences themselves are available from https://github.com/Shamir-Lab/SCAPP/tree/master/scapp/data.

A node in the assembly graph is annotated as containing a PSG hit (Table 1 step 1b) if there is a BLAST match between one of the PSG sequences and the sequence corresponding to the node ($\geq$ 75% sequence identity along $\geq$ 75% of the length of the gene).

Plasmid sequence score annotation

We use PlasClass [9] to annotate each node in the assembly graph with a plasmid score (Table 1 step 1c). PlasClass uses a set of logistic regression classifiers for sequences of different lengths to assign a classification score reflecting the likelihood of each node to be of plasmid origin.

We re-weight the node scores according to the sequence length as follows. For a given sequence of length L and plasmid probability p assigned by the classifier, the re-weighted plasmid score is: $s = 0.5 + \frac{p - 0.5}{1 + e^{-0.001(L-2000)}}$. This tends to pull scores towards 0.5 for short sequences, for which there is lower confidence, while leaving scores of longer sequences practically unchanged.

Long nodes ($L > 10$ kbp) with low plasmid score ($s < 0.2$) are considered probable chromosomal sequences and are removed, simplifying the assembly graph. Similarly, long nodes ($L > 10$ kbp) with high plasmid score ($s > 0.9$) are considered probable plasmid nodes.

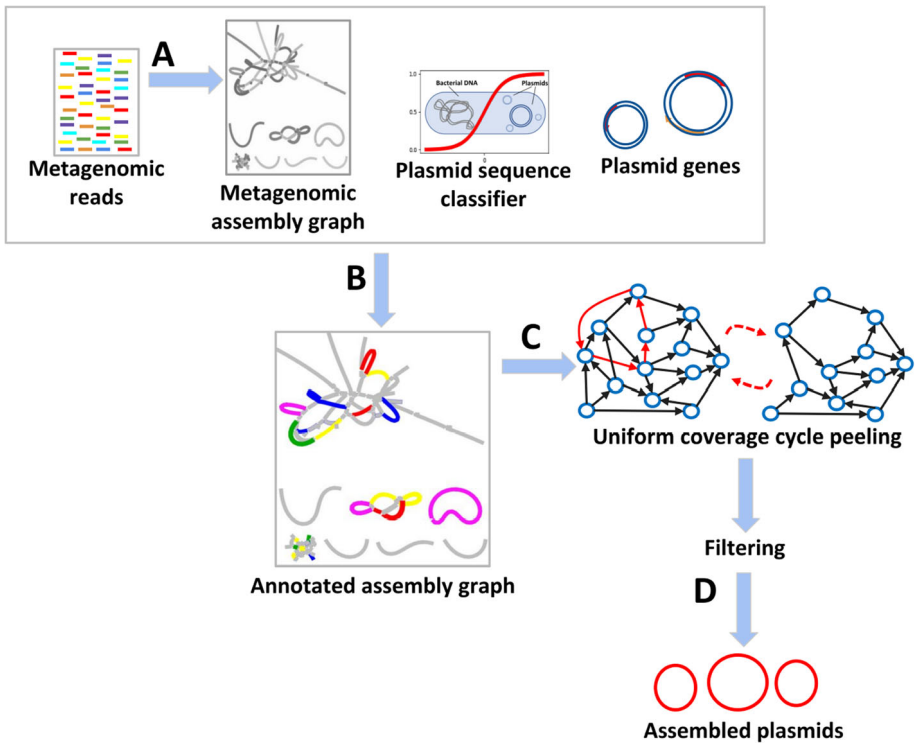


Fig. 1 Graphical overview of the SCAPP algorithm. (A) The metagenomic assembly graph is created from the sample reads. (B) The assembly graph is annotated with read mappings, presence of plasmid specific genes, and node weights based on sequence length, coverage, and plasmid classifier score. (C) Potential plasmids are iteratively peeled from the assembly graph. An efficient algorithm finds cyclic paths in the annotated assembly graph that have low weight and high chance of being plasmids. Cycles with uniform coverage are peeled. (D) Confident plasmid predictions are retained using plasmid sequence classification and plasmid-specific genes to remove likely false positive potential plasmids

Assigning node weights

In order to apply the peeling idea, nodes are assigned weights (Table 1 step 1d) so that lower weights correspond to higher likelihood to be assembled into a plasmid. Plasmid score and PSG annotations are incorporated into the node weights. A node with plasmid score s is assigned a weight $w(v) = (1-s)/(C \cdot L)$ where C is the depth of coverage of the node's sequence and L is the sequence length. This gives lower weight to nodes with higher coverage, longer sequence, and higher plasmid scores. Nodes with PSG hits are assigned a weight of zero, making them more likely to be integrated into any lowest-weight cycle in the graph that can pass through them.

Finding low-weight cycles in the graph

The core of the SCAPP algorithm is to iteratively find a lowest weight ("lightest") cycle going through each node in the graph for consideration as a potential plasmid. We use the bidirectional single-source, single-target shortest path implementation of the NetworkX Python package [12].

The order that nodes are considered matters since in each iteration potential plasmids are peeled from the graph, affecting the cycles that may be found in subsequent iterations. The plasmid annotations are used to decide the order that nodes are considered: first all nodes with PSGs, then all probable plasmid nodes, and then all other nodes in the graph (Table 1 step 2). If the lightest cycle going through a node meets certain criteria described below, it is peeled off, changing the coverage of nodes in the graph. Performing the search for light cycles in this order ensures that the cycles through more likely plasmid nodes will be considered before other cycles.

Assessing coverage uniformity

The lightest cyclic path, weighted as described above, going through each node is found and evaluated. Recycler sought a cycle with near uniform coverage, reasoning that all contigs that form a plasmid should have roughly the same coverage. However, this did not take into account the overlap of the cycle with other paths in the graph (see Fig. 2). To account for this, we instead compute a discounted coverage score for each node in the cycle based on its interaction with other paths as follows:

The *discounted coverage* of a node v in the cycle C is its coverage $cov(v)$ times the fraction of the coverage on all its

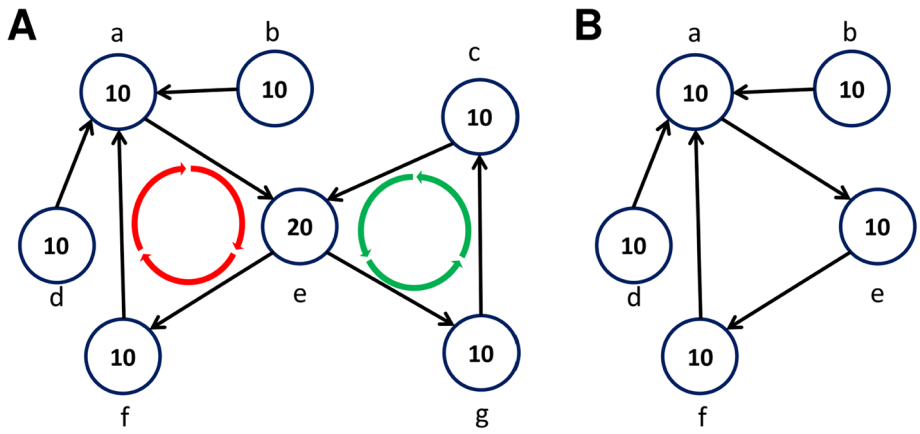


Fig. 2 Evaluating and peeling cycles. Numbers inside nodes indicate coverage. All nodes in the example have equal length. **A** Cycles (a, e, f) and (c, e, g) have the same average coverage (13.33) and coefficient of variation (CV, 0.35), but their discounted CV values differ: The discounted coverage of node a is 6, and the discounted coverage of node e is 10 in both cycles. The left cycle has discounted CV=0.22 and the right has discounted CV=0. By peeling off the mean discounted coverage of the right cycle (10) one gets the graph in **B**. Note that nodes g, c were removed from the graph since their coverage was reduced to 0, and the coverage of node e was reduced to 10

neighbors (both incoming and outgoing), $\mathcal{N}(v)$, that is on those neighbors that are in the cycle (see Fig. 2):

$$cov'(v, C) = cov(v) \cdot \left(\sum_{u \in C \wedge u \in \mathcal{N}(v)} cov(u) / \sum_{u \in \mathcal{N}(v)} cov(u) \right)$$

A node v in cycle C with contig length $len(v)$ is assigned a weight f corresponding to its fraction of the length of the cycle: $f(v, C) = len(v) / \sum_{u \in C} len(u)$. These weights are used to compute the weighted mean and standard deviation of the discounted coverage of the nodes in the cycle: $\mu_{cov'}(C) = \sum_{u \in C} f(u, C) cov'(u, C)$,

$$STD_{cov'}(C) = \sqrt{\sum_{u \in C} f(u, C)(cov'(u, C) - \mu_{cov'}(C))^2}$$

The coefficient of variation of C, which evaluates its coverage uniformity, is the ratio of the standard deviation to the mean:

$$CV(C) = \frac{STD_{cov'}(C)}{\mu_{cov'}(C)}$$

Finding potential plasmid cycles

After each lightest cycle has been generated, it is evaluated as a potential plasmid based on its structure in the assembly graph, the PSGs it contains, its plasmid score, paired-end read links, and coverage uniformity. The precise evaluation criteria are described in Additional file 1, Section S3. A cycle that passes them is defined as a potential plasmid (Table 1 steps 3–5). The potential plasmid cycles are peeled from the graph in each iteration as defined in Algorithm 2 (see also Fig. 2).

Filtering confident plasmid assemblies

In the final stage of SCAPP, PSGs and plasmid scores are used to filter out likely false-positive plasmids from the output and create a set of confident plasmid assemblies (Table 1 step 6). All potential plasmids are assigned a length-weighted plasmid score and are annotated with PSGs as was done for the contigs during graph annotation. Those that belong to at least two of the following sets are reported as confident plasmids: (a) potential plasmids containing a match to a PSG, (b) potential plasmids with plasmid score > 0.5, (c) self-loop nodes.

Results

We tested SCAPP on simulated metagenomes, human gut metagenomes, a human gut plasmidome dataset that we generated and also on parallel metagenome and plasmidome datasets from the same cow rumen microbiome specimen that we generated. The test settings and evaluation methods are described in Additional file 1, Section S5.

Simulated metagenomes

We created seven read datasets simulating metagenomic communities of bacteria and plasmids and assembled them. Datasets of increasing complexity were created as shown in Table 2. We randomly selected bacterial genomes along with their associated plasmids and used realistic distributions for genome abundance and plasmid copy number. Further details of the simulation can be found in Additional file 1, Section S4, and in Additional file 2. 5M paired-end reads were generated for Sim1 and

Table 2 Performance on simulated metagenome datasets. The number of covered plasmids (# covered) reports the number of the simulation plasmids that were covered by reads along at least 95% of their length. The set of covered plasmids is used as the gold standard in calculating the performance metrics. The numbers in parentheses are the median plasmid lengths (in kbp). F1 score is presented as a percent

				Recycler		mpSpades		SCAPP	
Sample	**# genomes**	**# plasmids**	**# covered**	**# plasmids**	**F1**	**# plasmids**	**F1**	**# plasmids**	**F1**
Sim1	10	9 (5.9)	9 (5.9)	7 (5.6)	50.0	1 (4.3)	20.0	5 (5.6)	57.1
Sim2	50	47 (19.3)	37 (13.5)	20 (3.8)	40.1	9 (5.0)	39.1	23 (5.5)	43.3
Sim3	200	210 (22.4)	136 (9.6)	61 (3.6)	32.8	27 (7.0)	32.3	48 (5.8)	42.9
Sim4	200	177 (25.4)	132 (12.7)	62 (4.1)	40.8	29 (6.0)	36.5	51 (6.2)	48.9
Sim5	300	318 (23.9)	253 (9.6)	115 (3.6)	35.2	53 (5.1)	33.8	100 (6.5)	47.5
Sim6	400	480 (13.5)	368 (9.1)	138 (3.0)	28.5	59 (5.5)	27.1	118 (5.5)	36.5
Sim7	500	571 (17.3)	410 (8.7)	132 (3.5)	31.1	69 (5.3)	28.1	141 (5.2)	40.5

Sim2, 10M for Sim3 and Sim4, and 20M for Sim5, Sim6, and Sim7.

Table 2 presents features of the simulated datasets and reports the performance of Recycler, mpSpades, and SCAPP on them. For brevity we report only F1 scores; precision and recall scores are reported in Supplementary Table 1, Additional file 1 (Section S6). Here, and throughout, all scores are adjusted to percent. SCAPP had the highest F1 score in all cases, followed by Recycler. SCAPP consistently achieved higher precision than Recycler, allowing it to perform better overall. mpSpades had the highest precision, but assembled far fewer plasmids than the other tools and gained lower recall and F1 scores. In fact, most of the plasmids assembled by mpSpades were also assembled by the other tools (see Figure S1 in Additional file 1), suggesting that these plasmids were easier to capture.

All of the tools assembled mostly shorter plasmids as reflected in the median plasmid lengths. This is likely due to the higher coverage and simplicity in the assembly graph of these plasmids, as also evidenced by the shorter lengths of the covered plasmids. SCAPP assembled many more long plasmids ($>$ 10 kbp) than the other tools, achieving much higher recall and higher F1 score for these longer plasmids than the other tools, at the cost of some precision (see Supplementary Table 2 in Additional file 1, Section S6 for results broken down by short and long plasmids).

Human gut microbiomes

We tested the plasmid assembly algorithms on data of twenty publicly available human gut microbiome samples selected from the study of Vrieze et al. [13]. The true set of plasmids in these samples is unknown. Instead, we matched all assembled contigs to PLSDB [14] and considered the set of the database plasmids that were covered by the contigs as the gold standard (see Additional file 1, Section S5 for details). All tools were evaluated according to the same gold standard. We note that this limits the evaluation to known plasmids, potentially overcounting the number of false positive plasmids. We chose the human gut microbiome in this experiment and the next, as it is one of the most widely studied microbiome environments so plasmids in gut microbiome samples are most likely to be represented in the database.

Table 3 presents the results of the three algorithms averaged across all twenty samples. The detailed results on each of the samples are presented in Supplementary Table 2 and Figure S2, Additional file 1 (Section S7). SCAPP performed best in more cases, with mpSpades failing to assemble any gold standard plasmid in over half the samples. We note that all of the cases where SCAPP had recall of 0 occurred when the number of gold standard plasmids was very small and the other tools also failed to assemble them. On the largest samples with the most gold standard plasmids SCAPP performed best, highlighting its superior performance on the types of samples most likely to be of interest in experiments aimed at plasmid assembly. SCAPP consistently outperformed Recycler by achieving higher precision, a result that is consistent with the other experiments.

Human gut plasmidome

The protocol developed in Brown Kav et al. [15] enables extraction of DNA from isolate or metagenomic samples

Table 3 Performance on the human gut metagenomes. Number of plasmids, the median plasmid length (in kbp), and performance measures for all tools are averaged across the twenty samples. The average number of plasmids and median length of the gold standard sets of plasmids were 4.8 and 12.4 respectively

Tool	**# plasmids**	**Median length**	**Precision**	**Recall**	**F1**
Recycler	15.9	3.6	7.1	36.4	10.9
mpSpades	6.5	5.0	7.9	17.4	10.3
SCAPP	9.8	4.4	11.5	36.4	16.1

with the plasmid content highly enriched. The sequence contents of such a sample is called the *plasmidome* of the sample. This enrichment for plasmid sequences increases the chance of revealing the plasmids in the sample. The protocol was assessed to achieve samples with at least 65% plasmid contents by Krawczyk et al. [5]. We sequenced the plasmidome of the human gut microbiome from a healthy adult male according to the plasmid enrichment protocol. 18,616,649 paired-end reads were sequenced with the Illumina HiSeq2000 platform, read length 150bp and insert size 1000.

The gold standard set of plasmids, determined as for the gut metagenome samples, consisted of 74 plasmids (median length = 2.1 kbp). Note that the plasmidome extraction process over-amplifies shorter plasmids, as reflected in the shorter median plasmid length. Performance was computed as in the metagenomic samples and is shown in Table 4. SCAPP achieved best overall performance, while mpSpades had lower precision and much lower recall than the other tools.

Notably, although the sample was obtained from a healthy donor, some of the plasmids reconstructed by SCAPP matched reference plasmids found in potentially pathogenic hosts such as *Klebsiella pneumoniae*, pathogenic serovars of *Salmonella enterica*, and *Shigella sonnei*. The detection of plasmids previously isolated from pathogenic hosts in the healthy gut indicates potential pathways for transfer of virulence genes.

We used MetaGeneMark [16] to find potential genes in the plasmids assembled by SCAPP. Two hundred ninety-four genes were found, and we annotated them with the NCBI non-redundant (nr) protein database using BLAST. Forty-six of the plasmids contained 170 (58%) genes with matches in the database ($>$ 90% sequence identity along $>$ 90% of the gene length), of which 77 (45%) had known functional annotations, which we grouped manually in Fig. 3A. There were six antibiotic and toxin (such as heavy metal) resistance genes, all on plasmids that were not in the gold standard set, highlighting SCAPP's ability to find novel resistance carrying plasmids. Sixty of the 77 genes (78%) with functional annotations had plasmid-associated functions: replication, mobilization, recombination, resistance, and toxin-antitoxin systems. Twenty-nine out of the 33 plasmids that contained functionally annotated genes (88%) contained at least one of these plasmid associated functions. This provides a strong indication that SCAPP succeeded in assembling true plasmids of the human gut plasmidome.

We also examined the hosts that were annotated for the plasmid genes and found that almost all of the plasmids with annotated genes contained genes with annotations from a variety of hosts, which we refer to here as "broad-range" (see Fig. 3B). Of the 40 plasmids with genes from annotated hosts, only 10 (25%) had genes with annotated hosts all within a single phylum. This demonstrates that these plasmids assembled and identified by SCAPP may be involved in one stage of transferring genes, such as the antibiotic resistance genes we detected, across a range of bacteria.

Table 4 Performance on the human gut plasmidome. Number of plasmids, the median plasmid length (in kbp), and performance measures for all tools

Tool	# plasmids	Median length	Precision	Recall	F1
Recycler	93	2.1	15.1	37.8	21.5
mpSpades	53	3.0	11.3	9.4	10.3
SCAPP	82	2.4	17.1	35.9	23.1

Parallel metagenomic and plasmidome samples

We performed two sequencing assays on the same cow rumen microbiome sample of a four-month old calf. In one subsample total DNA was sequenced. In the other, plasmid-enriched DNA was extracted as described in Brown Kav et al. [15] and sequenced (see Fig. 4). 27,127,784 paired-end reads were sequenced in the plasmidome, and 54,292,256 in the metagenome. Both were sequenced on the Illumina HiSeq2000 platform with read length 150bp and insert size 1000.

This parallel data enabled us to assess the plasmids assembled on the metagenome using the plasmidome, without resorting to PLSDB matches as the gold standard. Such assessment is especially useful for samples from non-clinical environments such as the cow rumen, as PLSDB likely under-represents plasmids in them.

Table 5 summarizes the results of the three plasmid discovery algorithms on both subsamples. mpSpades made the fewest predictions and Recycler made the most. To compare the plasmids identified by the different tools, we considered two plasmids to be the same if their sequences matched at $>$ 80% identity across $>$ 90% of their length. The comparison is shown in Figure S3, Additional file 1 (Section S8). In the plasmidome subsample, 50 plasmids were identified by all three methods. Seventeen were common to the three methods in the metagenome. In both subsamples, the Recycler plasmids included all or almost all of those identified by the other methods plus a large number of additional plasmids. In the plasmidome, SCAPP and Recycler shared many more plasmids than mpSpades and Recycler.

We also evaluated the results of the plasmidome and metagenome assemblies by comparison to PLSDB as was done for the human gut samples. The metagenome contained only one matching PLSDB reference plasmid, and none of the tools assembled it. The plasmidome had only seven PLSDB matches, and mpSpades, Recycler, and SCAPP had F1 scores of 2.86, 2.67, and 1.74, respectively. The low fraction of PLSDB matches out of the assem-

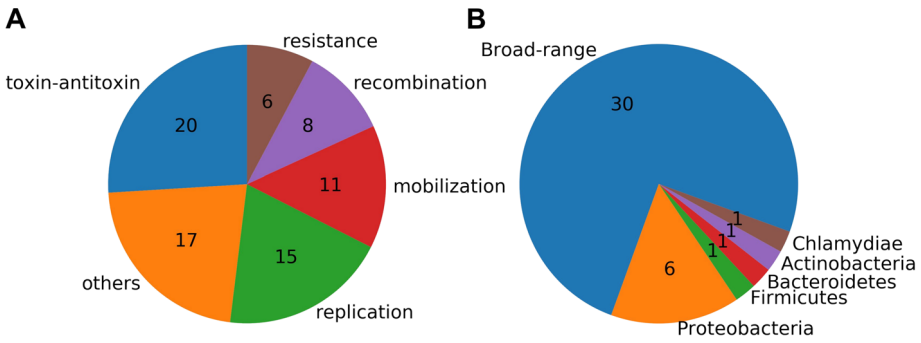


Fig. 3 Annotation of genes on the plasmids identified by SCAPP in the human gut plasmidome sample. **A** Functional annotations of the plasmid genes. **B** Host annotations of the plasmid genes. "Broad-range" plasmids had genes annotated with hosts from more than one phylum

bled plasmids suggests that the tools can identify novel plasmids that are not in the database.

In order to fully leverage the power of parallel samples, we computed the performance of each tool on the metagenomic sample using the reads of the plasmidomic sample, without doing any contig and plasmid assembly on the latter. The rationale was that the reads of the plasmidome represent the full richness of plasmids in the sample in a way that is not biased by a computational procedure or prior biological knowledge.

We calculated the *plasmidome read-based precision* by mapping the plasmidomic reads to the plasmids assembled from the metagenomic sample (Fig. 4). A plasmid with $>$ 90% of its length covered by more than one plasmidomic read was considered to be a true positive. The precision of an algorithm was defined as the fraction of true positive plasmids out of all reported plasmids. The *plasmidome read-based recall* was computed by mapping the plasmidomic reads to the contigs of the metagenomic assembly. Contigs with $>$ 90% of their length covered by plasmidomic reads at depth $>$ 1 were called *plasmidic contigs*. Plasmidic contigs that were part of the assembled plasmids were counted as true positives, and those that were not were considered false negatives. The recall was defined as the fraction of the plasmidic contigs' length that was integrated in the assembled plasmids. Note that

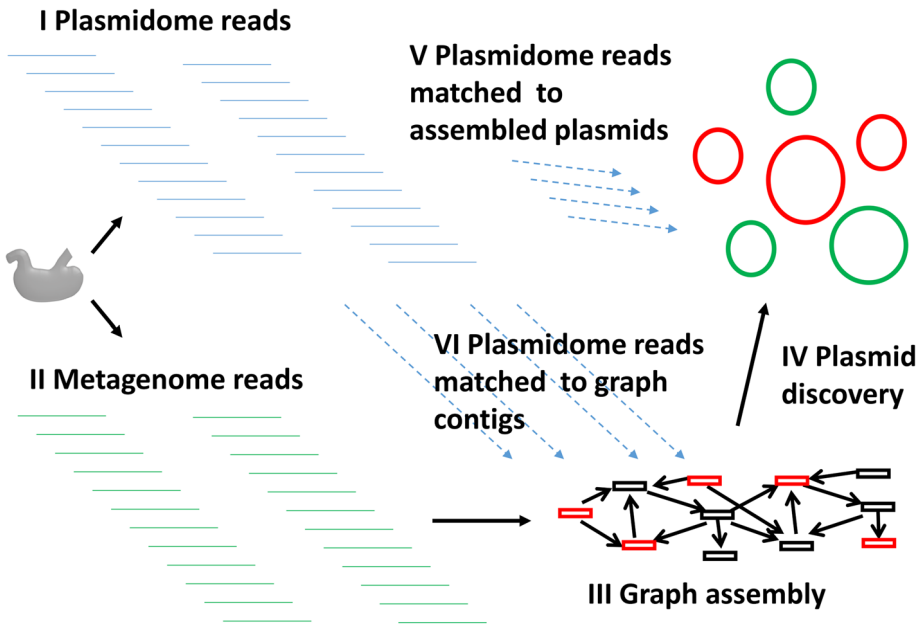


Fig. 4 Outline of the read-based performance assessment. Plasmidome (I) and metagenome reads (II) are obtained from subsamples of the same sample. (III) The metagenome reads are assembled into a graph. (IV) The graph is used to detect and report plasmids by the algorithm of choice. (V) The plasmidome reads are matched to assembled plasmids. Matched plasmids (red) are used to calculate plasmid read-based precision. (VI) The plasmidome reads are matched to the assembly graph contigs. Covered contigs (red) are considered plasmidic. The fraction of total length of plasmidic contigs included in the detected plasmids gives the plasmidome read-based recall

Table 5 Number of plasmids assembled by each tool and their median lengths (in kbp) for the parallel metagenome and plasmidome samples

Tool	metagenome		plasmidome	
	# plasmids	median length	# plasmids	median length
Recycler	60	4.3	147	1.7
SCAPP	25	5.8	110	1.8
mpSpades	26	6.2	65	2.0

the precision and recall here are measured using different units (plasmids and base pairs, respectively) so they are not directly related. For mpSpades, which does not output a metagenomic assembly, we mapped the contigs from the metaSPAdes assembly to the mpSpades plasmids using BLAST (> 80% sequence identity matches along > 90% of the length of the contigs).

There were 293 plasmidic contigs in the metagenome assembly graph, with a total length of 146.6 kbp. The plasmidome read-based performance is presented in Fig. 5A. All tools achieved a similar recall of around 12. SCAPP and mpSpades performed similarly, with SCAPP having slightly higher precision (24.0 vs 23.1) but slightly lower recall (11.9 vs 12.2). Recycler had a bit higher recall (13.1), but at the cost of far lower precision (11.7). Hence, a much lower fraction of the plasmids assembled by Recycler in the metagenome were actually supported by the parallel plasmidome sample, adding to the other evidence that the false positive rate of Recycler exceeds that of the other tools.

We also compared the plasmids assembled by each tool in the two subsamples. For each tool, we considered the plasmids it assembled from the plasmidome to be the gold standard set, and used it to score the plasmids it assembled in the metagenome. The results are shown in Fig. 5B. SCAPP had the highest precision. Since mpSpades had a much smaller gold standard set, it achieved higher recall and F1. Recycler output many more plasmids than the other tools in both samples, but had much lower precision, suggesting that many of its plasmid predictions may be spurious.

Next, we considered the union of the plasmids assembled across all tools as the gold standard set and recomputed the scores. We refer to them as "overall" scores. Figure 5C shows that overall precision scores were the same as in Fig. 5B, while overall recall was lower for all the tools, as expected. mpSpades underperformed because of its smaller set of plasmids, and SCAPP had the highest overall F1 score. Recycler performed relatively better on recall than the other tools as expected, as it reports many plasmids and has significant overlap with the plasmids reported by the other tools.

We detected potential genes in the plasmids assembled by SCAPP in the plasmidome sample and annotated them as we did for the human gut plasmidome. The gene function and host annotations are shown in Figure S4, Additional file 1 (Section S8). Out of 242 genes, only 34 genes from 17 of the plasmids had annotations, and only 18 of these had known functions, highlighting that many of the plasmids in the cow rumen plasmidome are as yet unknown. The high percentage of genes of plasmid function (15/18) indicates that SCAPP succeeded in assembling novel plasmids. Unlike in the human gut plasmidome, most of the plasmids with known host annotations had hosts from a single phylum.

Performance summary

We summarize the performance of the tools across all the test datasets in Table 6. The performance of two tools was

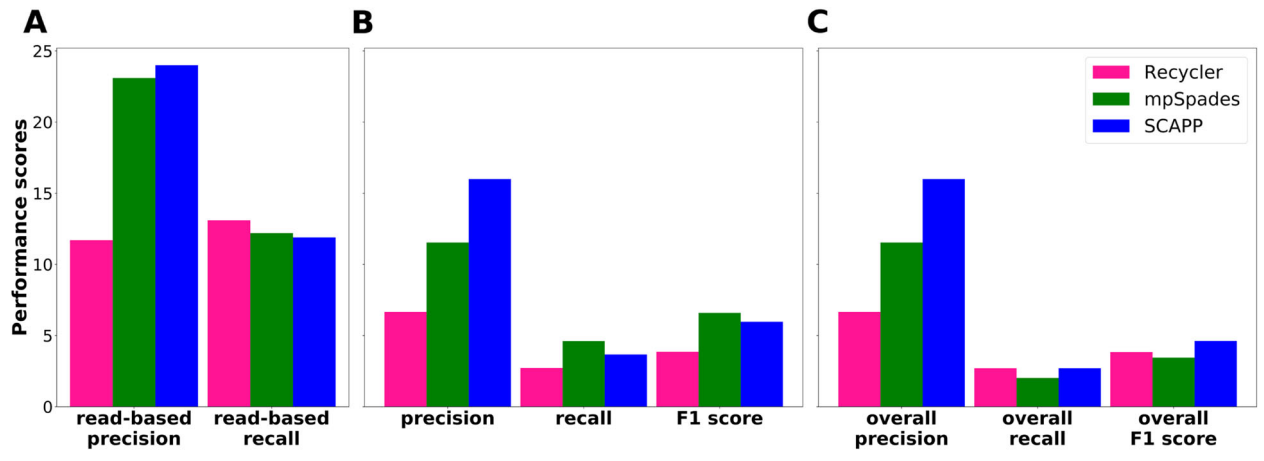


Fig. 5 Performance on the parallel datasets. **A** Plasmidome read-based performance. **B** Performance of each tool on the plasmids assembled from the metagenome using as gold standard the plasmids assembled from the plasmidome by the same tool. **C** Overall performance on the plasmids assembled from the metagenome compared to the union of all plasmids assembled by all tools in the plasmidome

Table 6 Summary of performance. Comparison of the performance of the tools on each of the datasets. When multiple samples were tested, the number of samples appears in parentheses, and average performance is reported. For the parallel samples results are for the evaluation of the metagenome based on the plasmidome, and precision and recall are plasmidome read-based. Unless otherwise stated, F1 score is used. Note that in the simulations, SCAPP $\gg$ mpSpades

Test	Ranking
Simulations (7)	SCAPP > Recycler > mpSpades
Human gut metagenomes (20)	SCAPP $\gg$ mpSpades > Recycler
Plasmidome	SCAPP > Recycler $\gg$ mpSpades
Parallel: within tool	mpSpades > SCAPP $\gg$ Recycler
Parallel: "overall", across tools	SCAPP > Recycler > mpSpades
Parallel: precision	SCAPP $\approx$ mpSpades $\gg$ Recycler
P arallel: recall	Recycler > mpSpades $\approx$ SCAPP

considered similar (denoted $\approx$) if their scores were within 5% of each other. Performance of one tool was considered to be much higher than the other ($\gg$) if its score was $>$ 30% higher (an increase of $5-30\%$ is denoted by $>$).

We see that in most cases SCAPP was the highest performer. Furthermore, in all other cases SCAPP performed close to the top performing tool.

Resource usage

The runtime and memory usage of the three tools are presented in Table 7. Recycler and SCAPP require assembly by metaSPAdes and pre-processing of the reads and the resulting assembly graph. SCAPP also requires post-processing of the assembled plasmids. mpSpades requires post-processing of the assembled plasmids with the plasmidVerify tool. The reported runtimes are for the full pipelines necessary to run each tool – from reads to assembled plasmids.

In almost all cases assembly was the most memory intensive step, and so all tools achieved very similar peak memory usage (within 0.01 GB). Therefore, we report the RAM usage for this step.

Table 7 Resource usage of the three methods. Peak RAM of the assembly step (metaSPAdes for Recycler and SCAPP, metaplasmidSPAdes for mpSpades) in GB. Runtime (wall clock time, in minutes) is reported for the entire pipeline including assembly and any pre-processing and post-processing required. Human metagenome results are an average across the 20 samples

		Runtime (minutes)		
Dataset	RAM (GB)	Recycler	mpSpades	SCAPP
Human metagenomes	21	115	103	130
Plasmidome	30	907	548	909
Parallel metagenome	148	2118	2132	2230
Parallel plasmidome	26	881	684	884

The assembly step was also the longest step in all cases. SCAPP was slightly slower than Recycler as a result of the additional annotation steps, and mpSpades was 5–40% faster. However, note that mpSpades does not output a metagenomic assembly graph, so users interested in both the plasmid and non-plasmid sequences in a sample would need to run metaSPAdes as well, practically doubling the runtime.

Performance measurements were made on a 44-core, 2.2 GHz server with 792 GB of RAM. Sixteen processes were used where possible. Recycler is single-threaded, so only one process was used for it.

Discussion

Plasmid assembly from metagenomic sequencing is a very difficult task, akin to finding needles in a haystack. This difficulty is demonstrated by the low numbers of plasmids found in real samples. Even in samples of the human gut microbiome, which is widely studied, relatively few plasmids that have matches in the extensive plasmid database PLSDB were recovered. Despite the challenges, SCAPP was able to assemble plasmids across a number of clinically relevant samples. SCAPP significantly outperformed mpSpades in simulation and on a range of human gut metagenome and plasmidome samples. In simulation mpSpades achieved very high precision at the expense of low recall, and SCAPP had higher combined F1 score. The high precision was not observed in real data, which is more difficult than the simulations. SCAPP was also consistently better than Recycler across almost all tests. Though SCAPP and Recycler share the idea of cycle peeling, SCAPP was shown to have higher precision, due to incorporating additional biological information and better edge weighting.

Another contribution of this study is the joint analysis of the parallel metagenome and plasmidome from the same sample. We show that this enables a novel way to evaluate plasmid assembly algorithms on the metagenome data, by using the coverage information from the plasmidome. This novel approach bypasses the need to rely on known plasmids for evaluation, which is biased due to research focus. We developed several evaluation metrics for such data, and think they can be useful for future plasmid studies, especially in non-clinical and non-human samples where plasmid knowledge is scarce.

A key difficulty in evaluation of performance of plasmid discovery algorithms is the lack of gold standard. The verification of reported plasmids is done either based on prior biological knowledge, which is biased, or by experimental verification, which is slow and expensive. Moreover, such verification evaluates precision but does not give information on the extent of missed plasmids, or recall. While

simulations can evaluate both parameters accurately, they are inherently artificial, and necessitate many modeling assumptions that are not fully supported by experimental data. For that reason we chose here to focus primarily on real data, and preferred diversity in the real data types over extensive but artificial simulations. The parallel samples strategy is another partial answer to this problem.

SCAPP has several limitations. Like the other de Bruijn graph-based plasmid assemblers, it may split a cycle into two when a shorter cycle is a sub-path of a longer cycle. It also has difficulties in finding very long plasmids, as these tend to not be completely covered and fragmented into many contigs in the graph. Note however that it produced longer cycles than Recycler. Compared to mpSpades, each algorithm produced longer cycles in different tests. Another limitation is the inherent bias in relying on known plasmid genes and plasmid databases, which tend to under-represent non-clinical samples. With further use of tools like SCAPP, perhaps with databases tailored to specific environments, further improvement is possible.

Conclusions

We introduced SCAPP, a new plasmid discovery tool based on combination of graph theoretical and biological considerations. Overall, SCAPP demonstrated better performance than Recycler and metaplasmidSpades in a wide range of real samples from diverse contexts. By applying SCAPP across large sets of samples, many new plasmid reference sequences can be assembled, enhancing our understanding of plasmid biology and ecology.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s40168-021-01068-z.

Additional file 1 — Supplementary information: Supplementary methods, experimental settings information, and results supporting the main text of this paper, including Figures S1-S4, Supplementary Tables 1 and 2. PDF file.

Additional file 2 — Simulation reference genomes: Tab-separated list of the human gut-specific reference genomes used in the simulations.

Acknowledgements

We thank members of the Shamir Lab for their help and advice—Roye Rozov, Lianrong Pu, Hagai Levi, and Nimrod Rappoport.

Authors' contributions

DP developed and implemented the SCAPP algorithm and benchmark experiments, performed analysis, and wrote the manuscript. AZ assisted with analysis and plasmid annotations and wrote the manuscript. MP curated plasmid-specific genes, assisted with gene annotations and wrote the manuscript. OF oversaw the parallel cow rumen metagenome-plasmidome experiment. AS oversaw the human gut plasmidome experiment. IM oversaw experimental and analysis aspects of the project and edited the manuscript. RS oversaw the computational and analysis aspects of project and edited the manuscript. All authors edited and approved the final manuscript.

Funding

PhD fellowships from the Edmond J. Safra Center for Bioinformatics at Tel-Aviv University and Israel Ministry of Immigrant Absorption (to DP). Israel Science Foundation (ISF) grant 1339/18, US - Israel Binational Science Foundation (BSF) and US National Science Foundation (NSF) grant 2016694 (to RS), ISF grant 1947/19 and ERC Horizon 2020 research and innovation program grant 640384 (to IM).

Availability of data and materials

The datasets supporting the conclusions of this article are available in the sequence read archive (SRA), accession numbers: ERR1297645, ERR1297651, ERR1297671, ERR1297685, ERR1297697, ERR1297700, ERR1297720, ERR1297738, ERR1297751, ERR1297770, ERR1297785, ERR1297796, ERR1297798, ERR1297810, ERR1297822, ERR1297824, ERR1297834, ERR1297838, ERR1297845, ERR1297852 (for the human gut metagenomes); accession SRR11038083 (for the human gut plasmidome); and accessions SRR11038085 and SRR11038085 (for the cow rumen metagenome and plasmidome samples, respectively). Project name: SCAPP
Project homepage: https://github.com/Shamir-Lab/SCAPP
Operating system: Platform independent (tested on Linux)
Programming language (Python3)
License: MIT

Declarations

Ethics approval and consent to participate

Sequencing of the human gut plasmidome was approved by the local ethics committee of Clalit HMO, approval number 0266-15-SOR. Extraction and sequencing of the cow rumen microbiome was approved by the local ethics committee of the Volcani Center, approval numbers 412/12IL and 566/15IL.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Blavatnik School of Computer Science, Tel Aviv University, 6997801 Tel Aviv, Israel. [2]Department of Life Sciences, Ben-Gurion University of the Negev and the National Institute for Biotechnology in the Negev, 8410501 Beer-Sheva, Israel. [3]Institute of Microbiology, University of Innsbruck, A-6020 Innsbruck, Austria. [4]Health Sciences, Ben-Gurion University of the Negev, 8410501 Beer-Sheva, Israel. [5]Soroka University Medical Center, 8410501 Beer-Sheva, Israel.

Received: 14 August 2020 Accepted: 1 April 2021
Published online: 25 June 2021

References

1. Arredondo-Alonso S, Willems R, van Schaik W, Schürch A. On the (im)possibility of reconstructing plasmids from whole-genome short-read sequencing data. Microb Genomics. 2017;3(10):000128.
2. Carattoli A, Zankari E, García-Fernández A, Larsen M, Lund O, Villa L, Aarestrup F, Hasman H. In silico detection and typing of plasmids using PlasmidFinder and plasmid multilocus sequence typing. Antimicrob Agents Chemother. 2014;58(7):3895–903.
3. Zhou F, Xu Y. cBar: a computer program to distinguish plasmid-derived from chromosome-derived sequence fragments in metagenomics data. Bioinforma. 2010;26(16):2051–2.
4. Arredondo-Alonso S, Bootsma M, Hein Y, Rogers MR, Corander J, Willems RJ, Schürch AC. gplas: a comprehensive tool for plasmid analysis using short-read graphs. Bioinformatics. 2020;36(12):3874–6.
5. Krawczyk P, Lipinski L, Dziembowski A. PlasFlow: predicting plasmid sequences in metagenomic data using genome signatures. Nucleic Acids Res. 2018;46(6):35.
6. Antipov D, Hartwick N, Shen M, Raiko M, Lapidus A, Pevzner P. plasmidSPAdes: assembling plasmids from whole genome sequencing data. Bioinforma. 2016;32(22):3380–7.

7. Rozov R, Brown Kav A, Bogumil D, Shterzer N, Halperin E, Mizrahi I, Shamir R. Recycler: an algorithm for detecting plasmids from de novo assembly graphs. Bioinforma. 2017;33(4):475–82.
8. Antipov D, Raiko M, Lapidus A, Pevzner P. Plasmid detection and assembly in genomic and metagenomic data sets. Genome Res. 2019;29(6):961–8.
9. Pellow D, Mizrahi I, Shamir R. PlasClass improves plasmid sequence classification. PLoS Comput Biol. 2020;16(4):1007781.
10. Li H. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. arXiv preprint arXiv:1303.3997. 2013.
11. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R. The sequence alignment/map format and samtools. Bioinforma. 2009;25(16):2078–9.
12. Hagberg A, Schult D, Swart P. Exploring network structure, dynamics, and function using NetworkX. In: Varoquaux G, Vaught T, Millman J, editors. Proceedings of the 7th Python in Science Conference (SciPy). Pasadena: Los Alamos National Lab (LANL); 2008. p. 11–5.
13. Vrieze A, Van Nood E, Holleman F, Salojärvi J, Kootte R, Bartelsman J, Dallinga–Thie G, Ackermans M, Serlie M, Oozeer R, et al. Transfer of intestinal microbiota from lean donors increases insulin sensitivity in individuals with metabolic syndrome. Gastroenterol. 2012;143(4):913–6.
14. Galata V, Fehlmann T, Backes C, Keller A. PLSDB: a resource of complete bacterial plasmids. Nucleic Acids Res. 2018;47(D1):195–202.
15. Brown Kav A, Benhar I, Mizrahi I. A method for purifying high quality and high yield plasmid dna for metagenomic and deep sequencing approaches. J Microbiol Meth. 2013;95(2):272–9.
16. Zhu W, Lomsadze A, Borodovsky M. Ab initio gene identification in metagenomic sequences. Nucleic Acids Res. 2010;38(12):132.

Publisher's Note

Supplementary information for SCAPP: An algorithm for improved plasmid assembly in metagenomes

David Pellow, Alvah Zorea, Maraike Probst, Ori Furman, Arik Segal, Itzhak Mizrahi, and Ron Shamir

January 10, 2021

S1 Alternatives for user set parameters

The SCAPP pipeline is highly flexible, and many of the options and parameters can be set by the user. In most cases, we recommend using the default options and settings. Some of the alternatives that can be chosen by the user are described below. All of the parameter settings that may be changed by the user are fully documented at: `https://github.com/Shamir-Lab/SCAPP`.

Read mapping: The user has the option of providing a sorted and indexed BAM alignment file created by any method.

Plasmid-specific genes: The user may add any set of PSGs or remove any of those included with SCAPP.

Plasmid classification scores: The sequences may be classified using PlasFlow and the PlasFlow classification output file can be provided to SCAPP.

Algorithm thresholds: Thresholds for finding plasmid gene matches, defining probable plasmid and chromosomal sequences, identifying potential plasmids, filtering them, and many more can all be user-defined. The full software documentation at `https://github.com/Shamir-Lab/SCAPP` details all of these user options.

S2 Plasmid-specific genes

We created four sets of plasmid-specific genes (PSGs) by database mining and expert curation:

1. MOB genes: 890 amino acid sequences of plasmid maintenance genes curated by plasmid biologists from the Mizrahi Lab (Ben-Gurion University) and filtered computationally (see details of filtering below).
2. Plasmid ORFs: 4276 nucleotide sequences corresponding to ORFs annotated with 'mobilization', 'conjugation', 'partitioning', 'toxin-antitoxin', 'replication', or 'recombination' from a large set of putative plasmids found by the Mizrahi Lab and then filtered computationally.
3. ACLAME plasmid genes: 4813 nucleotide sequences of genes that make up 96 gene families in the ACLAME database [1] that were manually selected as possibly plasmid-specific. The set of genes was deduplicated and filtered computationally.
4. PLSDB-specific ORFs: 94478 plasmid-specific sequences determined as follows: We used MetaGeneMark [2] to predict genes in the plasmid sequences from PLSDB (v.2018_12_05) [3]. We then counted the number of BLAST matches ($> 75\%$ identity match along $> 75\%$ of the gene length) to these genes in both PLSDB and bacterial reference genomes from NCBI (downloaded January 9, 2019). We

considered each predicted gene that appeared in the plasmids more than 20 times and was $> 20\times$ more prevalent in the plasmids than in the genomes to be plasmid-specific.

Sets 1–3 were filtered as follows: We counted matches between the sequences and PLSDB plasmids and NCBI bacterial reference genomes as for the PLSDB-specific ORFs (set 4). We excluded any gene that had more than 4 matches to bacterial genes *and* met one of the following conditions: (1) ≤ 4 matches to plasmid genes and $> 4\times$ as many matches to bacterial genes as plasmid genes; or, (2) > 4 plasmid gene matches, but $\leq 4\times$ as many matches to plasmid genes as to bacterial genes.

We did not search for and remove duplicate genes between sets, and did not back-translate the amino acid sequences.

S3 Potential plasmid cycle criteria

Once the set of lightest cycles has been generated, each cycle is evaluated as a potential plasmid based on its structure in the assembly graph, the PSGs it contains, its plasmid score, paired-end read links, and coverage uniformity. A cycle is defined as a potential plasmid if one of the following criteria is met:

1. The cycle is formed by an isolated "compatible" self-loop node v, i.e. $len(v) > 1000$, $indeg(v) = outdeg(v) = 1$, and at least one of the following conditions holds:
   (a) v has a high plasmid score $s(v) > 0.9$.
   (b) v has a PSG hit.
   (c) $< 10\%$ of the paired-end reads with a mate on v have the other mate on a different node.
2. The cycle is formed by a connected compatible self-loop node v, i.e. $len(v) > 1000$, $indeg(v) > 1$ or $outdeg(v) > 1$, and $< 10\%$ of the paired-end reads with a mate on v have the other mate on a different node.
3. The cycle is not formed by a self-loop and has:
   (a) Uniform coverage: $CV(C) < 0.5$, and
   (b) Consistent mate-pair links: a node in the cycle is defined as an "off-path dominated" node if the majority of the paired-end reads with one mate on the node have the other mate on a node that is not in the cycle. If less than half the nodes in the cycle are "off-path dominated", then we consider the mate-pair links to be consistent.

S4 Simulation of metagenomes with plasmids

To create the simulated metagenomes, we downloaded all completed whole genome bacterial reference sequences from RefSeq (RefSeq database updated on March 11, 2020). We first compiled a list of bacterial strains or species that have been previously identified as prevalent in the human gut from three sources: (1) The list compiled by Alneberg *et al.* [4] (Sup Table 1). (2) Species with abundance > 0.01 in at least one human gut sample from the human microbiome project (HMP1) [5] as estimated by MetaPhLan (abundance table available from `https://www.hmpdacc.org/HMSMCP/#data`). (3) The "dominant species" identified by Forster *et al.* [6] (Sup Table 5) in the HGG (Human Gastrointestinal Bacteria Genome Collection). We searched for the strains or species on this combined list in the RefSeq database, giving preference to strain level matches. When multiple references appeared (for example, when a listed species has multiple reference strains), we gave preference to those with longer plasmids (>10kbp), followed by those with any plasmid, choosing randomly between references with the same preference. The list of human gut specific bacteria used in the simulations contained 145 references, and is provided in Additional file 2.

For each simulation we first selected from the human gut specific bacteria and then supplemented with randomly selected reference sequences to reach the desired number of genomes. Since the plasmids sequenced with completed whole bacterial genomes are usually long, we also supplemented with a fixed number of shorter

(<10kbp) plasmids, selected randomly and associated at random with host genomes in the simulation. (5 short plasmids were added in Sim1, 15 in Sim2, 50 in Sim3 and Sim4, 100 in Sim5, 150 in Sim6, and 200 in Sim7.)

Genome abundance and plasmid copy number were assigned using realistic distributions. For genome abundance we used the log-normal distribution ($\mu = 1.5$, $\sigma = 1$), normalized so that the relative abundances sum to 1. This long-tailed distribution mimics the abundance distribution of real microbiome samples. Plasmids were assigned the same abundance as their hosts, and plasmid copy number was assigned according to one of several geometric distributions according to the plasmid length. The parameter of the geometric distribution of a plasmid of length L was set to be

$$p = \begin{cases} log_{10}(L)/30, & 1\text{kbp} \leq L < 10\text{kbp} \\ log_{10}(L)/20, & 10\text{kbp} \leq L < 100\text{kbp} \\ log_{10}(L)/10, & 100\text{kbp} \leq L < 1\text{Mbp} \\ 1, & L \geq 1\text{Mbp} \end{cases}$$

This makes it more likely for shorter plasmids to have higher copy numbers, in accordance with observed plasmid copy number patterns.

Paired-end short reads were simulated from the genome references using InSilicoSeq [7] with the HiSeq error model (default read length = 126bp). To reflect circularity of the plasmids and bacterial genomes, multiple copies of the reference sequence were concatenated before generating reads.

S5 Experimental settings and evaluation

All metagenomes were assembled using the SPAdes assembler (v3.13) with the `--meta` option. The default of 16 threads were used, and the maximum memory was set to 750 GB. metaplasmidSPAdes (mpSpades) was run with the same parameters. mpSpades internally chooses the maximal value of k to use for the k-mer length in the assembly graph. We matched the values of k used in SPAdes to these values for each dataset. Defaults were used for all other options for Recycler and SCAPP. In practice, the maximum k value was 77 for the simulations and human metagenomic samples, and 127 for the plasmidome and parallel metagenome-plasmidome samples.

For a simulated metagenome, the set of reference plasmids included in the simulation that were covered along $> 95\%$ of their length by simulated reads was used as the gold standard. Reads were mapped using BWA [8], and coverage at each base of the reference plasmids was called using bedtools [9].

We used BLAST to match the assembled plasmids to the gold standard plasmid sequences. A plasmid assembled by one of the tools was considered to be a true positive if $> 90\%$ of its length was covered by BLAST matches to $> 90\%$ of a reference with $> 80\%$ sequence identity. The rest of the assembled plasmids were considered to be false positives. Gold standard plasmids that did not have assembled plasmids matching them were considered to be false negatives. Precision was defined as $TP/(TP + FP)$ and recall was defined as $TP/(TP + FN)$, where TP, FP, and FN were the number of true positive, false positive, and false negative plasmids, respectively. The F1 score was defined as the harmonic mean of precision and recall. Precision, recall, and F1 values are adjusted to percentage throughout. Note that these metrics evaluate the complete assembly of entire plasmids, and do not capture rates of local misassembly or short range assembly errors within each assembled plasmid.

For the human microbiome and plasmidome samples, the set of plasmids serving as the gold standard was selected from PLSDB (v.2018_12_05) [3], a large curated plasmid database. After filtering duplicate plasmids, the PLSDB contains 13469 reference plasmids. The contigs from the metaSPAdes assembly were matched against the plasmids in PLSDB using BLAST. Matches between a contig and a reference plasmid with sequence identity $> 85\%$ were marked and a contig was said to match a reference if $> 85\%$ of its length was marked. Reference plasmids with $> 90\%$ of their lengths covered by marked regions of the matching contigs were used as the gold standard.

Table 1 Full performance on simulated metagenome datasets. The gold standard is the number of plasmids in the simulation that are covered by simulated reads (# covered).

		Recycler			mpSpades			SCAPP		
Sample	# covered	precision	recall	F1	precision	recall	F1	precision	recall	F1
Sim1	9	57.1	44.4	50.0	100	11.1	20.0	80.0	44.4	57.1
Sim2	37	60	32.4	40.1	100	24.3	39.1	56.5	35.1	43.3
Sim3	136	52.5	23.9	32.8	96.3	19.4	32.3	81.3	29.1	42.9
Sim4	132	62.9	30.2	40.8	100	22.3	36.5	86.3	34.1	48.9
Sim5	253	55.6	25.7	35.2	96.2	20.5	33.8	77.2	34.3	47.5
Sim6	368	51.4	19.7	28.5	96.6	15.8	27.1	72.9	24.4	36.5
Sim7	410	62.9	20.6	31.1	95.7	16.5	28.1	75.9	27.6	40.5

The set of plasmids assembled by each method was compared to the gold standard set using BLAST. A predicted plasmid was considered a true positive if there were sequence matches at $> 80\%$ identity between the plasmid and a gold standard plasmid that covered more than 90% of their lengths.

Note that in the case of the real samples, if two assembled plasmids matched to the same reference gold standard plasmid sequence(s), then one of them was considered to be a false positive. This strict definition penalized methods for unnecessarily splitting potential plasmid genomes into multiple different plasmids. If there were multiple gold standard reference plasmids that were matched to a single assembled plasmid, then none of them was considered as a false negative. The precision, recall, and F1 score were calculated as for the simulation.

For the parallel metagenome-plasmidome sample, plasmidomic reads were aligned to the plasmid sequences and metagenome assembly contigs using BWA [8]. Coverage at each base of each metagenomic contig was called using bedtools [9].

To compare the overlap between plasmids identified by the different tools, we considered two plasmids to be the same if their sequences matched at $> 80\%$ identity across $> 90\%$ of their length. For visualization purposes, when two plasmids in one tool match one plasmid in another, they are represented as one overlap in the venn diagram (Figures S1 and S3).

S6 Extended results for simulated datasets

Table 1 reports the full precision, recall, and F1 performance results for all tools on the simulated metagenome datasets. Table 2 reports the performance results when split by length into shorter (< 10 kbp) and longer (≥ 10 kbp) plasmids. Figure S1 shows the overlap between the plasmids assembled by each tool in the simulated metagenomes.

Table 2 Performance on simulated metagenome datasets stratified by short plasmids (< 10 kbp) and long plasmids (≥ 10 kbp). The number of gold-standard plasmids (GS) for each length bin is indicated in parentheses.

		Recycler				mpSpades				SCAPP			
Sample	Length bin (# GS)	# plasmids	precision	recall	F1	# plasmids	precision	recall	F1	# plasmids	precision	recall	F1
Sim1	< 10 kb (5)	5	80.0	80.0	80.0	1	100.0	20.0	33.3	4	100.0	80.0	88.9
	≥ 10 kb (4)	2	0.0	0.0	0.0	0	-	0.0	0.0	1	0.0	0.0	0.0
Sim2	< 10 kb (18)	18	66.7	66.7	66.7	7	100.0	38.9	56.0	17	70.6	66.7	68.6
	≥ 10 kb (19)	2	50.0	5.3	9.5	2	50.0	5.3	9.5	6	33.3	10.5	16.0
Sim3	< 10 kb (69)	55	50.9	41.8	45.9	22	95.5	31.3	47.2	37	89.2	49.3	63.5
	≥ 10 kb (67)	6	66.7	8.0	11.0	5	100.0	7.5	13.9	11	54.5	9.0	15.4
Sim4	< 10 kb (63)	57	61.4	58.3	59.8	25	100.0	41.0	58.1	42	90.5	62.3	73.8
	≥ 10 kb (69)	5	80.0	5.8	10.8	4	100.0	5.8	11.0	9	66.7	8.8	15.6
Sim5	< 10 kb (128)	106	56.6	48.4	52.2	48	95.8	37.1	53.5	80	90.0	58.5	70.9
	≥ 10 kb (125)	9	44.4	3.2	6.0	5	100.0	4.0	7.7	30	43.3	10.4	16.8
Sim6	< 10 kb (196)	125	52.0	34.6	41.5	51	96.1	25.8	40.7	88	87.5	42.3	57.0
	≥ 10 kb (172)	13	46.2	3.5	6.5	8	100.0	4.7	8.9	30	33.3	5.8	10.0
Sim7	< 10 kb (225)	121	61.2	34.1	43.8	61	95.1	26.9	41.9	109	86.2	46.5	60.5
	≥ 10 kb (185)	11	81.8	4.9	9.2	8	87.5	3.8	7.3	32	40.6	7.1	12.0

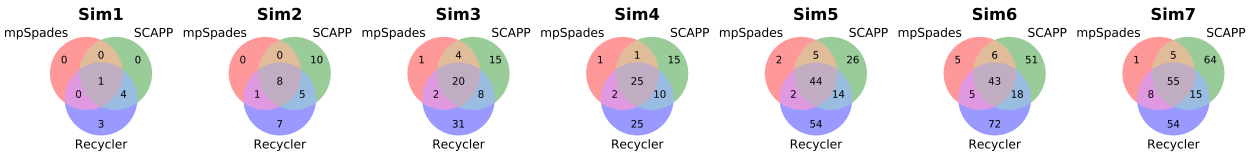


Figure S1 Plasmid overlap between tools in simulation. Overlap of the plasmids assembled by the tools on each of the simulated metagenomes.

S7 Extended results for human metagenomes

Figure S2 presents the F1 scores of the plasmid assemblers across all human gut metagenome samples. Table 3 reports the full results and the number of plasmids assembled by each tool and the median plasmid length for each of the human gut microbiome samples.

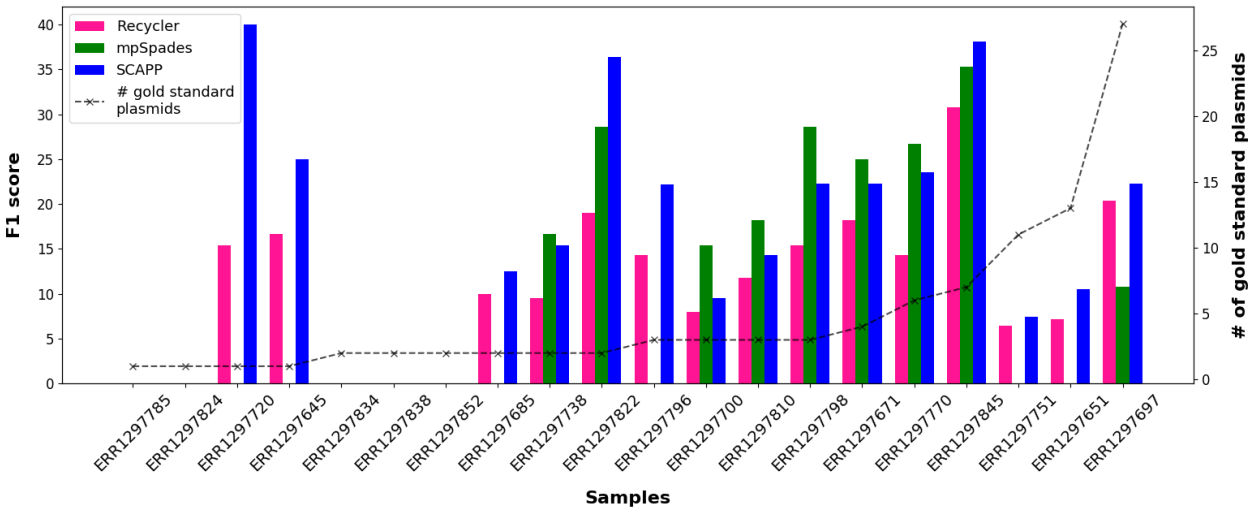


Figure S2 Results on 20 human gut metagenomes. F1 scores of the plasmids assembled by Recycler, mpSpades and SCAPP in the human gut microbiome samples (accessions given on x-axis), calculated using PLSDB plasmids as the gold standard. The dashed line shows the number of gold standard plasmids in each sample. Where bars are omitted the F1 score was 0.

Table 3 Full results on the human gut microbiome samples and number of plasmids and median lengths (in kbp).

	Gold standard	Recycler				mpSpades				SCAPP			
Sample	# plasmids (median length)	# plasmids (median length)	Precision	Recall	F1	# plasmids (median length)	Precision	Recall	F1	# plasmids (median length)	Precision	Recall	F1
ERR1297785	1 (6.0)	14 (2.4)	0	0	0	4 (4.7)	0	0	0	8 (4.3)	0	0	0
ERR1297824	1 (6.0)	15 (3.2)	0	0	0	6 (5.2)	0	0	0	8 (4.8)	0	0	0
ERR1297720	1 (2.7)	12 (3.4)	8.3	100.0	15.4	3 (4.2)	0	0	0	4 (3.4)	25.0	100.0	40.0
ERR1297645	1 (2.7)	11 (3.2)	9.1	100.0	16.7	7 (5.2)	0	0	0	7 (4.5)	14.3	100.00	25.0
ERR1297834	2 (4.1)	5 (4.4)	0	0	0	3 (6.4)	0	0	0	3 (6.3)	0	0	0
ERR1297838	2 (6.5)	17 (2.0)	0	0	0	4 (4.6)	0	0	0	8 (5.4)	0	0	0
ERR1297852	2 (19.5)	17 (5.1)	0	0	0	5 (5.3)	0	0	0	6 (5.2)	0	0	0
ERR1297685	2 (2.5)	18 (5.2)	5.6	50.0	10.0	7 (6.1)	0	0	0	14 (4.3)	7.1	50.0	12.5
ERR1297738	2 (19.5)	19 (5.1)	5.3	50.0	9.5	10 (4.9)	10.0	50.0	16.7	11 (5.1)	9.1	50.0	15.4
ERR1297822	2 (2.2)	19 (4.4)	10.5	100.0	19.0	5 (4.5)	20.0	50.0	28.6	9 (4.1)	22.2	100.0	36.4
ERR1297796	3 (8.9)	11 (2.9)	9.1	33.3	14.3	5 (3.6)	0	0	0	6 (2.8)	16.7	33.3	22.2
ERR1297700	3 (5.6)	22 (2.9)	4.5	33.3	8.0	10 (4.4)	10.0	33.3	15.4	18 (4.4)	5.6	33.3	9.5
ERR1297810	3 (5.6)	14 (3.5)	7.1	33.3	11.8	8 (4.4)	12.5	33.3	18.2	11 (4.6)	9.1	33.3	14.3
ERR1297798	3 (8.9)	11 (2.9)	9.1	50.0	15.4	5 (6.4)	20.0	50.0	28.6	7 (2.9)	14.3	50.0	22.2
ERR1297671	4 (7.4)	8 (3.8)	12.5	33.3	18.2	5 (4.2)	20.0	33.3	25.0	6 (4.0)	16.7	33.3	22.2
ERR1297770	6 (8.9)	23 (3.4)	8.7	40.0	14.3	10 (4.7)	20.0	40.0	26.7	12 (4.4)	16.7	40.0	23.5
ERR1297845	7 (8.9)	20 (3.3)	20.0	66.7	30.8	11 (5.9)	27.3	50.0	35.3	15 (3.8)	26.7	66.7	38.1
ERR1297751	11 (4.9)	20 (4.0)	5.0	9.1	6.5	6 (5.6)	0	0	0	16 (4.4)	6.3	9.1	7.4
ERR1297651	13 (114.2)	15 (3.2)	6.7	7.7	7.1	4 (4.9)	0	0	0	6 (4.5)	16.7	7.7	10.5
ERR1297697	27 (2.1)	25 (3.8)	20.0	20.8	20.4	11 (5.4)	18.2	7.7	10.8	21 (4.5)	23.8	20.8	22.2

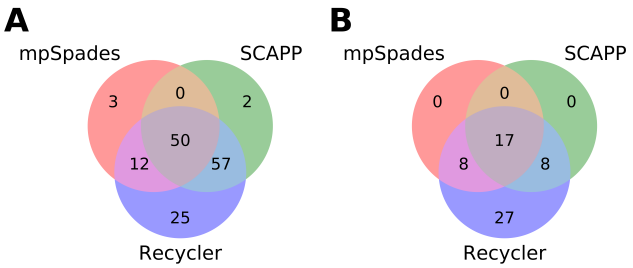


Figure S3 Number of plasmids assembled by each tool on the parallel samples. A: Plasmidome sample. B: Metagenome sample. Discrepancies between the numbers in the diagram and Table 4 are due to cases of overlaps between two plasmids in one tool to one plasmid in another, which were counted as one.

S8 Extended results for parallel plasmidome-metagenome

Figure S3 shows the overlap between the plasmids assembled by the tools in the parallel cow rumen plasmidome and metagenome samples.

Figure S4 shows the annotations of the gene functions and hosts for the plasmids assembled in the rumen plasmidome.

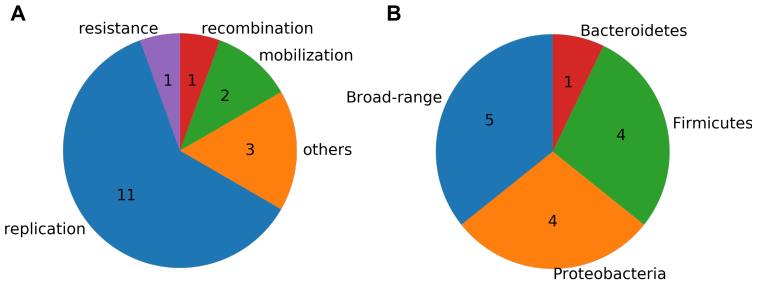


Figure S4 Annotation of genes on the plasmids identified by SCAPP in the rumen plasmidome sample. A: Functional annotations of the plasmid genes. B: Host annotations of the plasmid genes.

References

[1] Leplae, R., Lima-Mendez, G., Toussaint, A.: ACLAME: a classification of mobile genetic elements, update 2010. Nucleic Acids Research **38**(suppl_1), 57–61 (2009)

[2] Zhu, W., Lomsadze, A., Borodovsky, M.: Ab initio gene identification in metagenomic sequences. Nucleic Acids Research **38**(12), 132–132 (2010)

[3] Galata, V., Fehlmann, T., Backes, C., Keller, A.: PLSDB: a resource of complete bacterial plasmids. Nucleic Acids Research **47**(D1), 195–202 (2018)

[4] Alneberg, J., Bjarnason, B.S., De Bruijn, I., Schirmer, M., Quick, J., Ijaz, U.Z., Lahti, L., Loman, N.J., Andersson, A.F., Quince, C.: Binning metagenomic contigs by coverage and composition. Nature methods **11**(11), 1144–1146 (2014)

[5] Methé, B.A., Nelson, K.E., Pop, M., Creasy, H.H., Giglio, M.G., Huttenhower, C., Gevers, D., Petrosino, J.F., Abubucker, S., Badger, J.H., *et al.*: A framework for human microbiome research. nature **486**(7402), 215 (2012)

[6] Forster, S.C., Kumar, N., Anonye, B.O., Almeida, A., Viciani, E., Stares, M.D., Dunn, M., Mkandawire, T.T., Zhu, A., Shao, Y., *et al.*: A human gut bacterial genome and culture collection for improved metagenomic analyses. Nature biotechnology **37**(2), 186–192 (2019)

[7] Gourlé, H., Karlsson-Lindsjö, O., Hayer, J., Bongcam-Rudloff, E.: Simulating illumina metagenomic data with insilicoseq. Bioinformatics **35**(3), 521–522 (2018)

[8] Li, H.: Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. arXiv preprint arXiv:1303.3997 (2013)

[9] Quinlan, A.R., Hall, I.M.: Bedtools: a flexible suite of utilities for comparing genomic features. Bioinformatics **26**(6), 841–842 (2010)

Chapter 5

Discussion

In this thesis I described my work on efficient methods for analysis of high throughput sequencing data. I first introduced the universal hitting set problem and DOCKS, our heuristic algorithm to generate a small UHS that could be used to define a low density minimizer scheme. Next, I presented PlasClass, an efficient plasmid sequence classifier with state-of-the-art classification performance. Finally, I presented SCAPP, a metagenomic plasmid assembler that uses biological knowledge about plasmid sequences and genes to improve precision of plasmid assembly in an efficient manner. In the latter two works, I implemented the algorithms as publicly available open-source software tools and benchmarked their performance against existing methods.

Below I discuss each of the papers, their impact, and possible future directions for each work.

5.1 UHS and improved selection schemes

Our paper on the UHS problem, and the follow-up paper defining low density minimizer orders gave rise to a number of other works trying to improve on solutions to the UHS problem and on the density of minimizer orders.

Ekim et al. [31] improved the runtime of DOCKS and introduced PASHA, a parallel randomized algorithm based on set cover approximation to more efficiently construct UHS that are only slightly larger than those constructed by DOCKS.

Marçais et al.[72] proved that asymptotically in k the optimal UHS is of size $|\Sigma|^k/w$ and thus there is a minimizer scheme compatible with this UHS that has density asymptotically approaching $o(1/w)$ as $k \to \infty$. They provided a construction for such a UHS and minimizer scheme. ReMuval [27] naively extended UHS generated by DOCKS to larger k and iteratively pruned each extension using an efficient ILP formulation. This allowed for lower density minimizers for larger k. Zheng et al. [136] bounded the length of the maximum remaining path length in the de Bruijn graph after removing the Mykkeltveit decycling set. Miniception [135] constructs UHS that are $(2 + o(1))$-approximations of the optimum for larger k that are essentially closed syncmers.

Despite the interest in this area, there are still open questions and more work to be done in this field. The best performing UHS constructions and minimizer orders are inefficient and do not scale to larger k, and achieving small UHS for larger k and w is an unmet challenge. There are likely still constructions to create smaller UHS and better lower bounds on the size of the optimal solution. It may even be possible to provably achieve the optimal solution, at least in some interesting cases.

Recent work on syncmers has called into question the focus on density and on window guarantees, suggesting that conservation under mutation and error is a more important property for k-mer selection schemes to optimize. The subset of k-mers selected by 1-local schemes such as syncmers could likely be optimized for conservation and density jointly, either in expectation or on real genomic sequences. Other metrics are also relevant in some tasks, and the work on AdaOrder [35] suggests directly optimizing selection schemes for a specific task on specific sequences.

In our own work to improve the efficiency of UHS construction we are now experimenting with neural network models trained on existing UHS that can be iteratively applied to larger k. These methods still do not improve running times enough to achieve much larger k. In fact, in that work we observed that a minimizer scheme based only on the Mykkeltveit decycling set may be both of low density and efficient to compute, and we plan to explore this in the future.

5.2 Plasmid sequence classification

In PlasClass we improved upon state-of-the-art plasmid classification using a simple logistic regression model. This work demonstrated that extant neural network models trained on k-mer features do not achieve best performance on this task. Rather, the training data is much more important. To facilitate continual improvement in plasmid sequence classification it will be necessary to have standard well-curated plasmid sequence databases. Plasmid classification tools that are easily or automatically re-trained as plasmid sequence databases grow over time are necessary to maintain and improve performance.

One main challenge we faced in our work was the lack of established benchmarks on which to compare plasmid classification tools. We created test simulations using realistic distributions for bacterial abundance and plasmid copy number, but future work could expand these efforts to create simulations that match distributions directly learned from data. When using realistic distributions, the class imbalance between plasmid and bacterial contigs creates a challenge in evaluating classifiers; even with a very low false positive rate a classifier may end up with low precision.

Some other works "balance" the class sizes instead of using realistically imbalanced classes and thus report results that are not representative of performance on real data. Short contigs also presented a key challenge in our work on PlasClass as they have less representative k-mer profiles. We trained separate models for different length contigs to address this issue. As shorter contigs are much more abundant than longer in real data, poor performance on these more difficult sequences can drastically reduce classifier performance. Most tools, including PlasClass, filter out the shortest contigs and do not classify them. Raising the length threshold can result in inflated performance and it is difficult to compare results between tools that use different thresholds. Additionally, some works reported length-weighted (i.e. per base) performance, that also inflates performance by giving higher weight to the more easily classified longer sequences. For these reasons as well, it is critical to have standard benchmarks and established performance criteria for evaluating plasmid classification tools.

As long read technologies, such as those offered by PacBio and Oxford Nanopore Technologies, become more widely used for metagenomic sequencing, plasmid classifiers can be adapted to work on this data. In many cases long reads will be as long as

contigs assembled using short-read data and individual reads can be classified using the same techniques as currently used for assembled contigs. High accuracy long reads also raise the possibility of more easily sequencing or assembling full plasmid genomes, which could allow classifying plasmid sequences by aligning full genomes to existing databases or searching for sets of plasmid marker genes in the sequences.

While PlasClass and many other tools only use compositional features (i.e. k-mer profiles), some newer tools such as plasmidVerify and PlasX demonstrate the importance of gene content features. Binning using assembly graph structure and coverage information, possibly across multiple samples, has also improved the performance of plasmid classifications. Future directions should incorporate all of these features to achieve even better plasmid sequence classification. Another promising direction is the introduction of multi-way classifiers. These ideas would all apply to metagenomic long read classification as well. Ultimately, the goal will be to efficiently and accurately identify all sequences in a metagenomic sample as originating from bacteria, plasmid, archaea, virus, fungi and other eukaryotic genome sequences.

5.3 Plasmid assembly

In SCAPP we improved on previous metagenomic plasmid assemblers by more carefully assessing coverage uniformity and incorporating extensive biological knowledge about plasmid sequence contents. This biological knowledge was crucial for reducing false positive and increasing precision of the tool. Another promising approach was offered by DomCycle [111], which used a very principled statistical analysis of read coverage and orientation to identify potential plasmid cycles. Finally, binning individual plasmids has been used in isolate but in metagenomic samples it was only used for classification so far. Given the prevalence and success of binning for bacterial genome assembly in metagenomic samples, it could likely be beneficial for plasmid assembly as well. Methods such as GraphPlas [127] could be modified to bin individual plasmid OTUs rather than just binary classification. Future work on metagenomic plasmid assembly should probably combine all the advantages above.

One challenge in creating more complex and better performing plasmid assemblers will be efficiently scaling them to larger and larger datasets. Current tools are limited in their ability to assemble plasmids from very large metagenomes or pooled samples. The initial metagenomic assembly can also be a bottleneck. A possible

future direction is to pre-filter large read sets (or metagenomic assembly graphs) to reduce computational overhead. Future plasmid assemblers should also focus on software engineering for efficiency in addition to any methodological advances. It is also important that users understand when it is necessary to perform the computationally heavy task of assembly, and when a less intensive task such as sequence classification or mapping to a database of known references could be sufficient.

As in the case of plasmid sequence classification, plasmid assembly lacks good benchmark datasets and standard evaluations. The CAMI 2 benchmark included circular sequences, but they are arguably not reflective of plasmid sequences and distributions in most real metagenomic samples. In our work we also contributed a novel evaluation method and dataset. Our collaborators at the Mizrahi lab sequenced the same cow rumen sample both using a standard metagenomic sequencing protocol and using a plasmidome sequencing protocol they developed that filters out chromosomal DNA before sequencing. This allowed us to evaluate the performance of metagenomic plasmid assembly by determining what fraction of contigs covered by plasmidome reads were assembled into plasmids ("recall") and what fraction of the assembled plasmids were actually covered by plasmidome reads ("precision"). While many tool developers would not be able to generate such a dataset, it would be worthwhile to construct a standard realistic benchmark. One possibility would be to construct a synthetic sample mixing at least 100 bacterial species carrying known plasmids with known copy numbers at known abundances. The widespread adoption of such a benchmark could drive the development of even better plasmid assemblers.

Plasmid assembly has the potential to be greatly improved by long read sequencing technologies, which are increasingly being used on metagenomic samples. Very long reads could capture entire plasmids in a single read, removing the need for any graph based assembly or binning methods. Assembly graphs constructed from long read datasets are much simpler and more easily resolved than short read assembly graphs, making it likely that cyclic paths representing plasmids could be found more easily. Plasmid assembly from long reads opens the door to new challenges that could not yet be approached using only short reads. Specifically, some current methods in long read metagenomic assembly attempt to disambiguate sequences at the strain level. Luo et al. [67] developed a tool for viral haplotyping to resolve viral strain genomes from viral mixtures. Bickhart et al. [11] use Hi-C and high accuracy HiFi long reads to resolve bacterial strains from metagenomic samples. Similar methods

could be developed for plasmid assembly, allowing them to be studied at the strain level and opening new avenues to understand plasmid evolution and ecology.

Bibliography

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.

[2] W. B. Andreopoulos, A. M. Geller, M. Lucke, et al. Deeplasmid: deep learning accurately separates plasmids from bacterial chromosomes. *Nucleic Acids Research*, 50(3):e17–e17, 2022.

[3] D. Antipov, N. Hartwick, M. Shen, et al. PlasmidSPAdes: assembling plasmids from whole genome sequencing data. *Bioinformatics*, 32(22):3380–3387, 2016.

[4] D. Antipov, M. Raiko, A. Lapidus, and P. A. Pevzner. Plasmid detection and assembly in genomic and metagenomic data sets. *Genome Research*, 29(6):961–968, 2019.

[5] D. Antipov, M. Raiko, A. Lapidus, and P. A. Pevzner. Metaviral spades: assembly of viruses from metagenomic data. *Bioinformatics*, 36(14):4126–4129, 2020.

[6] S. Arredondo-Alonso, M. Bootsma, Y. Hein, et al. gplas: a comprehensive tool for plasmid analysis using short-read graphs. *Bioinformatics*, 36(12):3874–3876, 2020.

[7] S. Arredondo-Alonso, M. R. Rogers, J. C. Braat, et al. mlplasmids: A user-friendly tool to predict plasmid-and chromosome-derived sequences for single species. *Microbial Genomics*, 4(11), 2018.

[8] S. Arredondo-Alonso, R. J. Willems, W. van Schaik, and A. C. Schürch. On the (im)possibility of reconstructing plasmids from whole-genome short-read sequencing data. *Microbial Genomics*, 3(10):e000128, 2017.

[9] A. Bankevich, S. Nurk, D. Antipov, et al. SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012. PMID: 22506599.

[10] J.-M. Belton, R. P. McCord, J. H. Gibcus, et al. Hi–C: A comprehensive technique to capture the conformation of genomes. *Methods*, 58(3):268–276, 2012.

[11] D. M. Bickhart, M. Kolmogorov, E. Tseng, et al. Generating lineage-resolved, complete metagenome-assembled genomes from complex microbial communities. *Nature Biotechnology*, 40(5):711–719, 2022.

[12] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[13] B. Bonev and G. Cavalli. Organization and function of the 3d genome. *Nature Reviews Genetics*, 17(11):661–678, 2016.

[14] K. M. Boycott, M. R. Vanstone, D. E. Bulman, and A. MacKenzie. Rare-disease genetics in the era of next-generation sequencing: discovery to translation. *Nature Reviews Genetics*, 14:681–691, 2013.

[15] F. P. Breitwieser, J. Lu, and S. L. Salzberg. A review of methods and databases for metagenomic classification and assembly. *Briefings in Bioinformatics*, 20(4):1125–1136, 2017.

[16] J. A. Briggs, C. Weinreb, D. E. Wagner, et al. The dynamics of gene expression in vertebrate embryogenesis at single-cell resolution. *Science*, 360, 2018.

[17] P. D. Browne, W. Kot, T. S. Jørgensen, and L. H. Hansen. The mobilome: Metagenomic analysis of circular plasmids, viruses, and other extrachromosomal elements. *Methods in Molecular Biology*, 2075:253–264, 2020.

[18] A. L. Byrd, Y. Belkaid, and J. A. Segre. The human skin microbiome. *Nature Reviews Microbiology*, 16:143–155, 2018.

[19] A. Carattoli, E. Zankari, A. García-Fernández, et al. In silico detection and typing of plasmids using PlasmidFinder and plasmid multilocus sequence typing. *Antimicrobial Agents and Chemotherapy*, 58(7):3895–3903, 2014.

[20] J. H. Chase, J. T. Fouquier, M. Zare, et al. Geography and location are the primary drivers of office microbiome composition. *mSystems*, 1, 2016.

[21] Y. Che, Y. Xia, L. Liu, et al. Mobile antibiotic resistome in wastewater treatment plants revealed by nanopore metagenomic sequencing. *Microbiome*, 7, 2019.

[22] R. Chikhi, A. Limasset, and P. Medvedev. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

[23] C. Y. Chiu and S. A. Miller. Clinical metagenomics. *Nature Reviews Genetics*,

20(6):341–355, 2019.

[24] L. C. Conteville and A. C. P. Vicente. A plasmid network from the gut microbiome of semi-isolated human groups reveals unique and shared metabolic and virulence traits. *Scientific Reports*, 12, 2021.

[25] L. Coombe, V. Nikolić, J. Chu, I. Birol, and R. L. Warren. ntJoin: Fast and lightweight assembly-guided scaffolding using minimizer graphs. *Bioinformatics*, 36(12):3885–3887, 2020.

[26] N. G. De Bruijn. A combinatorial problem. In *Proc. Koninklijke Nederlandse Academie van Wetenschappen*, volume 49, pages 758–764, 1946.

[27] D. DeBlasio, F. Gbosibo, C. Kingsford, and G. Marçais. Practical universal k-mer sets for minimizer schemes. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, BCB '19, page 167–176, New York, NY, USA, 2019. Association for Computing Machinery.

[28] S. Deorowicz, M. Kokot, S. Grabowski, and A. Debudaj-Grabysz. KMC 2: fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015.

[29] R. Edgar. Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. *PeerJ*, 9:e10805, 2021.

[30] B. Ekim, B. Berger, and R. Chikhi. Minimizer-space de bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell Systems*, 2021.

[31] B. Ekim, B. Berger, and Y. Orenstein. A randomized parallel algorithm for efficiently finding near-optimal universal hitting sets. In *Research in Computational Molecular Biology*, pages 37–53. Springer International Publishing, 2020.

[32] M. Erbert, S. Rechner, and M. Müller-Hannemann. Gerbil: a fast and memory-efficient k-mer counter with GPU-support. *Algorithms for Molecular Biology*, 12(1):9, 2017.

[33] Z. Fang, J. Tan, S. Wu, et al. PPR-Meta: a tool for identifying phages and plasmids from metagenomic fragments using deep learning. *GigaScience*, 8(6):giz066, 2019.

[34] N. Fierer. Embracing the unknown: disentangling the complexities of the soil microbiome. *Nature Reviews Microbiology*, 15:579–590, 2017.

[35] D. Flomin, D. Pellow, and R. Shamir. Data set-adaptive minimizer order

reduces memory usage in k-mer counting. *Journal of Computational Biology*, 2022.

[36] M. V. Francia, A. Varsaki, M. P. Garcillán-Barcia, et al. A classification scheme for mobilization regions of bacterial plasmids. *FEMS Microbiology Reviews*, 28 1:79–100, 2004.

[37] C. Gawad, W. Koh, and S. R. Quake. Single-cell genome sequencing: current state of the science. *Nature Reviews Genetics*, 17(3):175–188, 2016.

[38] J. A. Gilbert, M. J. Blaser, J. G. Caporaso, et al. Current understanding of the human microbiome. *Nature Medicine*, 24(4):392–400, 2018.

[39] J. A. Gilbert and B. Stephens. Microbiology of the built environment. *Nature Reviews Microbiology*, 16(11):661–670, 2018.

[40] S. Golbabapour, M. A. Abdulla, and M. Hajrezaei. A concise review on epigenetic regulation: insight into molecular mechanisms. *International Journal of Molecular Sciences*, 12(12):8661–8694, 2011.

[41] R. Gomi, K. L. Wyres, and K. E. Holt. Detection of plasmid contigs in draft genome assemblies using customized Kraken databases. *Microbial Genomics*, 7(4), 2021.

[42] M. Hoang, H. Zheng, and C. Kingsford. DeepMinimizer: A differentiable framework for optimizing sequence-specific minimizer schemes. In *Research in Computational Molecular Biology*, pages 52–69. Springer International Publishing, 2022.

[43] G. Holley and P. Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de bruijn graphs. *Genome Biology*, 21(1):1–20, 2020.

[44] D. Holoch and D. Moazed. RNA-mediated epigenetic regulation of gene expression. *Nature Reviews Genetics*, 16(2):71–84, 2015.

[45] R. M. Idury and M. S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2(2):291–306, 1995. PMID: 7497130.

[46] L. Irber, P. T. Brooks, T. Reiter, et al. Lightweight compositional analysis of metagenomes with FracMinHash and minimum metagenome covers. *bioRxiv*, 2022.

[47] C. Jain, A. Dilthey, S. Koren, S. Aluru, and A. M. Phillippy. A fast approximate algorithm for mapping long reads to large reference databases. In *Research in Computational Molecular Biology*, pages 66–81. Springer International Publishing, 2017.

[48] C. Jain, A. Rhie, N. F. Hansen, S. Koren, and A. M. Phillippy. Long-read map-

ping to repetitive reference sequences using Winnowmap2. *Nature Methods*, pages 1–6, 2022.

[49] C. Jain, A. Rhie, H. Zhang, et al. Weighted minimizer sampling improves long read mapping. *Bioinformatics*, 36(Supp. 1):i111–i118, 2020.

[50] A. B. Kav, R. Rozov, D. Bogumil, et al. Unravelling plasmidome distribution and interaction with its hosting microbiome. *Environmental Microbiology*, 2019.

[51] D. Kim, L. Song, F. P. Breitwieser, and S. L. Salzberg. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome Research*, 26(12):1721–1729, 2016.

[52] D. C. Koboldt, K. M. Steinberg, D. E. Larson, R. K. Wilson, and E. R. Mardis. The next-generation sequencing revolution and its impact on genomics. *Cell*, 155:27–38, 2013.

[53] E. V. Koonin. Does the central dogma still stand? *Biology Direct*, 7(1):1–7, 2012.

[54] K. Kopotsa, J. O. Sekyere, and N. M. Mbelle. Plasmid evolution in carbapenemase-producing Enterobacteriaceae: a review. *Annals of the New York Academy of Sciences*, 1457, 2019.

[55] P. S. Krawczyk, L. Lipinski, and A. Dziembowski. PlasFlow: predicting plasmid sequences in metagenomic data using genome signatures. *Nucleic Acids Research*, 46(6):e35, 2018.

[56] K. Krishnan, T. Chen, and B. J. Paster. A practical guide to the oral microbiome and its relation to health and disease. *Oral Diseases*, 23 3:276–286, 2017.

[57] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):1–10, 2009.

[58] V. F. Lanza, M. de Toro, M. P. Garcillán-Barcia, et al. Plasmid flux in Escherichia coli ST131 sublineages, analyzed by plasmid constellation network (PLACNET), a new method for plasmid reconstruction from whole genome sequences. *PLoS Genetics*, 10(12):e1004766, 2014.

[59] D. Li, C.-M. Liu, R. Luo, K. Sadakane, and T.-W. Lam. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics*, 31(10):1674–1676, 2015.

[60] H. Li. Minimap and miniasm: fast mapping and de novo assembly for noisy

long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[61] H. Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018.

[62] H. Li. New strategies to improve minimap2 alignment accuracy. *arXiv preprint arXiv:2108.03515*, 2021.

[63] Y. Li and Xifeng Yan. MSPKmerCounter: A fast and memory efficient approach for k-mer counting. *arXiv preprint*, 2015.

[64] F. Ling, R. Whitaker, M. W. LeChevallier, and W.-T. Liu. Drinking water microbiome assembly induced by water stagnation. *The ISME journal*, 12(6):1520–1531, 2018.

[65] G. A. Logsdon, M. R. Vollger, and E. E. Eichler. Long-read human genome sequencing and its applications. *Nature Reviews Genetics*, pages 1–18, 2020.

[66] R. Luo, B. Liu, Y. Xie, et al. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. *GigaScience*, 1(1), 2012. 2047-217X-1-18.

[67] X. Luo, X. Kang, and A. Schönhuth. Strainline: full-length de novo viral haplotype reconstruction from noisy long reads. *Genome Biology*, 23(1):1–27, 2022.

[68] B. Ma, L. J. Forney, and J. Ravel. Vaginal microbiome: rethinking health and disease. *Annual Review of Microbiology*, 66:371–89, 2012.

[69] F. Maguire, B. Jia, K. L. Gray, et al. Metagenome-assembled genome binning methods with short reads disproportionately fail for plasmids and genomic islands. *Microbial genomics*, 6(10), 2020.

[70] U. Manber and G. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[71] A. Mane, M. Faizrahnemoon, and C. Chauve. A mixed integer linear programming algorithm for plasmid binning. In *RECOMB International Workshop on Comparative Genomics*, pages 279–292. Springer, 2022.

[72] G. Marçais, D. DeBlasio, and C. Kingsford. Asymptotically optimal minimizers schemes. *Bioinformatics*, 34(13):i13–i22, 2018.

[73] G. Marçais, D. Pellow, D. Bork, et al. Improving the performance of minimizers and winnowing schemes. *Bioinformatics*, 33(14):i110–i117, 2017.

[74] G. Marçais, B. Solomon, R. Patro, and C. Kingsford. Sketching and sublinear data structures in genomics. *Annual Review of Biomedical Data Science*, 2(1):93–118, 2019.

[75] C. Marchet, M. Kerbiriou, and A. Limasset. BLight: efficient exact associative structure for k-mers. *Bioinformatics*, 37(18):2858–2865, 2021.

[76] E. R. Mardis. The impact of next-generation sequencing technology on genetics. *Trends in Genetics*, 24(3):133–141, 2008.

[77] M. Meng, Y. Li, and H. Yao. Plasmid-mediated transfer of antibiotic resistance genes in soil. *Antibiotics*, 11, 2022.

[78] P. Menzel, K. L. Ng, and A. Krogh. Fast and sensitive taxonomic classification for metagenomics with Kaiju. *Nature Communications*, 7(1):1–9, 2016.

[79] F. Meyer, A. Fritz, Z.-L. Deng, et al. Critical assessment of metagenome interpretation: the second round of challenges. *Nature Methods*, 19(4):429–440, 2022.

[80] A. Mikheenko, A. Prjibelski, V. Saveliev, D. Antipov, and A. Gurevich. Versatile genome assembly evaluation with QUAST-LG. *Bioinformatics*, 34(13):i142–i150, 2018.

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

[82] W. K. Mousa, F. Chehadeh, and S. Husband. Recent advances in understanding the structure and function of the human microbiome. *Frontiers in Microbiology*, 13, 2022.

[83] R. Müller and C. Chauve. HyAsP, a greedy tool for plasmids identification. *Bioinformatics*, 35(21):4436–4439, 2019.

[84] N. Nagarajan and M. Pop. Sequence assembly demystified. *Nature Reviews Genetics*, 14(3):157–167, 2013.

[85] S. Nurk, D. Meleshko, A. Korobeynikov, and P. A. Pevzner. metaSPAdes: a new versatile metagenomic assembler. *Genome Research*, 27(5):824–834, 2017.

[86] J. Nyström-Persson, G. Keeble-Gagnère, and N. Zawad. Compact and evenly distributed k-mer binning for genomic sequences. *Bioinformatics*, 2021. btab156.

[87] B. D. Ondov, T. J. Treangen, P. Melsted, et al. Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17(1):1–14, 2016.

[88] Y. Orenstein, D. Pellow, G. Marçais, R. Shamir, and C. Kingsford. Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. *PLoS Computational Biology*, 13(10):e1005777, 2017.

[89] A. Orlek, N. Stoesser, M. F. Anjum, et al. Plasmid classification in an era

of whole-genome sequencing: Application in studies of antibiotic resistance epidemiology. *Frontiers in Microbiology*, 8, 2017.

[90] R. Ounit, S. Wanamaker, T. J. Close, and S. Lonardi. CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics*, 16(1):1–13, 2015.

[91] D. Pellow, A. Dutta, and R. Shamir. Using syncmers improves long-read mapping. *bioRxiv*, 2022.

[92] D. Pellow, I. Mizrahi, and R. Shamir. PlasClass improves plasmid sequence classification. *PLoS Computational Biology*, 16(4):e1007781, 2020.

[93] D. Pellow, A. Zorea, M. Probst, et al. SCAPP: an algorithm for improved plasmid assembly in metagenomes. *Microbiome*, 9(1):1–12, 2021.

[94] 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.

[95] L. Pradier, T. Tissot, A.-S. Fiston-Lavier, and S. Bedhomme. PlasForest: a homology-based random forest classifier for plasmid detection in genomic datasets. *BMC Bioinformatics*, 22(1):1–17, 2021.

[96] L. M. Proctor, H. H. Creasy, J. M. Fettweis, et al. The integrative human microbiome project. *Nature*, 569(7758):641–648, 2019.

[97] L. Pu and R. Shamir. 3CAC: improving the classification of phages and plasmids in metagenomic assemblies using assembly graphs. *bioRxiv*, 2022.

[98] P. H. Rampelotto, A. F. R. Sereia, L. F. V. de Oliveira, and R. Margis. Exploring the hospital microbiome by high-resolution 16S rRNA profiling. *International Journal of Molecular Sciences*, 20, 2019.

[99] M. Rautiainen and T. Marschall. MBG: Minimizer-based sparse de Bruijn Graph construction. *Bioinformatics*, 37(16):2476–2478, 2021.

[100] J. Ravel, P. Gajer, Z. Abdo, et al. Vaginal microbiome of reproductive-age women. *Proceedings of the National Academy of Sciences*, 108:4680 – 4687, 2010.

[101] J. Ren, N. A. Ahlgren, Y. Y. Lu, J. A. Fuhrman, and F. Sun. VirFinder: a novel k-mer based tool for identifying viral sequences from assembled metagenomic data. *Microbiome*, 5(1):1–20, 2017.

[102] 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.

[103] J. Robertson and J. H. Nash. MOB-suite: software tools for clustering, reconstruction and typing of plasmids from draft assemblies. *Microbial Genomics*, 4(8), 2018.

[104] J. Rodríguez-Beltrán, J. DelaFuente, R. Leon-Sampedro, R. C. MacLean, and A. San Millan. Beyond horizontal gene transfer: the role of plasmids in bacterial evolution. *Nature Reviews Microbiology*, 19(6):347–359, 2021.

[105] M. Roosaare, M. Puustusmaa, M. Möls, M. Vaher, and M. Remm. PlasmidSeeker: identification of known plasmids from bacterial whole genome sequencing reads. *PeerJ*, 6:e4588, 2018.

[106] W. P. Rowe. When the levee breaks: a practical guide to sketching algorithms for processing the flood of genomic data. *Genome Biology*, 20(1):1–12, 2019.

[107] R. Rozov, A. Brown Kav, D. Bogumil, et al. Recycler: an algorithm for detecting plasmids from de novo assembly graphs. *Bioinformatics*, 33(4):475–482, 2017.

[108] K. Sahlin. Flexible seed size enables ultra-fast and accurate read alignment. *bioRxiv*, 2022.

[109] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85, 2003.

[110] O. Schwengers, P. Barth, L. Falgenhauer, et al. Platon: identification and characterization of bacterial plasmid contigs in short-read draft assemblies exploiting protein sequence-based replicon distribution scores. *Microbial Genomics*, 6(10), 2020.

[111] N. Shalon, D. A. Relman, and E. Yaffe. Precise genotyping of circular mobile elements from metagenomic data uncovers human-associated plasmids with recent common ancestors. *Genome Research*, 2022.

[112] J. Shaw and Y. W. Yu. Theory of local k-mer selection with applications to long-read alignment. *Bioinformatics*, 2021. btab790.

[113] M. Shintani, Z. K. Sanchez, and K. Kimbara. Genomics of microbial plasmids: classification and identification based on replication and transfer systems and host taxonomy. *Frontiers in Microbiology*, 6, 2015.

[114] J. Sielemann, K. Sielemann, B. Brejová, T. Vinař, and C. Chauve. plASgraph - using graph neural networks to detect plasmid contigs from an assembly graph. *bioRxiv*, 2022.

[115] J. T. Simpson, K. Wong, S. D. Jackman, et al. ABySS: a parallel assembler

for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.

[116] R. K. Singh, H. W. Chang, D. Yan, et al. Influence of diet on the gut microbiome and implications for human health. *Journal of Translational Medicine*, 15, 2017.

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

[118] W. W. Soon, M. Hariharan, and M. P. Snyder. High-throughput sequencing for biology and medicine. *Molecular Systems Biology*, 9(1):640, 2013.

[119] R. D. Stewart, M. D. Auffret, A. Warr, et al. Compendium of 4,941 rumen metagenome-assembled genomes for rumen microbiome biology and enzyme discovery. *Nature Biotechnology*, 37:953 – 961, 2019.

[120] S. R. Stockdale, R. S. Harrington, A. N. Shkoporov, et al. Metagenomic assembled plasmids of the human microbiome vary across disease cohorts. *Scientific Reports*, 12, 2022.

[121] S. G. Tringe and E. M. Rubin. Metagenomics: DNA sequencing of environmental samples. *Nature Reviews Genetics*, 6(11):805–814, 2005.

[122] D. Usoskin, A. Furlan, S. Islam, et al. Unbiased classification of sensory neuron types by large-scale single-cell rna sequencing. *Nature Neuroscience*, 18:145–153, 2015.

[123] L. van der Graaf-Van Bloois, J. A. Wagenaar, and A. L. Zomer. RFPlasmid: predicting plasmid sequences from short-read assembly data using machine learning. *Microbial Genomics*, 7(11), 2021.

[124] J. C. Venter, K. Remington, J. F. Heidelberg, et al. Environmental genome shotgun sequencing of the Sargasso sea. *Science*, 304(5667):66–74, 2004.

[125] A.-C. Villani, R. Satija, G. Reynolds, et al. Single-cell RNA-seq reveals new types of human blood dendritic cells, monocytes, and progenitors. *Science*, 356, 2017.

[126] R. R. Wick, L. M. Judd, C. L. Gorrie, and K. E. Holt. Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLoS Computational Biology*, 13(6):e1005595, 2017.

[127] A. Wickramarachchi and Y. Lin. GraphPlas: Refined classification of plasmid sequences using assembly graphs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 19(1):57–67, 2022.

[128] D. E. Wood, J. Lu, and B. Langmead. Improved metagenomic analysis with Kraken 2. *Genome Biology*, 20(1):1–13, 2019.

[129] D. E. Wood and S. L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):1–12, 2014.

[130] S. H. Ye, K. J. Siddle, D. J. Park, and P. C. Sabeti. Benchmarking metagenomics tools for taxonomic classification. *Cell*, 178(4):779–794, 2019.

[131] M. K. Yu, E. C. Fogarty, and A. M. Eren. The genetic and ecological landscape of plasmids in the human gut. *bioRxiv*, 2022.

[132] Y. Yuan, C. Y.-L. Chung, and T.-F. Chan. Advances in optical mapping for genomic research. *Computational and Structural Biotechnology Journal*, 18:2051–2062, 2020.

[133] Y. Yue, H. Huang, Z. Qi, et al. Evaluating metagenomics tools for genome binning with real metagenomic datasets and CAMI datasets. *BMC Bioinformatics*, 21(1):1–15, 2020.

[134] D. R. Zerbino. Using the Velvet de novo assembler for short-read sequencing technologies. *Current Protocols in Bioinformatics*, 31(1):11–5, 2010.

[135] H. Zheng, C. Kingsford, and G. Marçais. Improved design and analysis of practical minimizers. *Bioinformatics*, 36(Supp. 1):i119–i127, 2020.

[136] H. Zheng, C. Kingsford, and G. Marçais. Lower density selection schemes via small universal hitting sets with short remaining path length. In *Research in Computational Molecular Biology*, pages 202–217. Springer International Publishing, 2020.

[137] H. Zheng, C. Kingsford, and G. Marçais. Sequence-specific minimizers via polar sets. *Bioinformatics*, 37(Supp 1):i187–i195, 2021.

[138] A. Zhernakova, A. Kurilshikov, M. J. Bonder, et al. Population-based metagenomics analysis reveals markers for gut microbiome composition and diversity. *Science*, 352:565 – 569, 2016.

[139] F. Zhou and Y. Xu. cBar: a computer program to distinguish plasmid-derived from chromosome-derived sequence fragments in metagenomics data. *Bioinformatics*, 26(16):2051–2052, 2010.

המטאגנומית עם הקריאות בדגימה הפלסמידית מאפשרת לנו למדוד כמה מהתוכן הפלסמידי הצליח האלגוריתם להרכיב (recall) וכמה מהקונטיגים שהורכבו הם באמת פלסמידים (precision).

השווינו את הביצועים של SCAPP ל-Recycler ו-metaplasmidSPAdes על מטאגנומים בסימולציות, על דגימות אמיתיות מהמיקרוביום של המעי האנושי, ועל פלסמידום של המעי האנושי שיצרנו. הביצועים של SCAPP עלו על אלו של Recycler ושל-metaplasmidSPAdes במרבית ההשוואות. הוא גם הצליח להרכיב פלסמידים חדשים בעלי עניין קליני בדגימות הפלסמידום של המעי האנושי.

SCAPP זמין כחבילת Python קלה לשימוש המאפשרת הרכבה של רצפי פלסמיד מלאים מדגימות מטאגנומיות. הכלי ניתן להורדה מ: github.com/Shamir-Lab/SCAPP. העבודה פורסמה בתור:

Pellow, D., Zorea, A., Probst, M., Furman, O., Segal, A., Mizrahi, I., & Shamir, R. (2021). **SCAPP: an algorithm for improved plasmid assembly in metagenomes.** *Microbiome*, *9*(1), 1-12.

מהגרף הינה UHS. אנחנו מראים שבפועל DOCKS מצליח ליצור UHS-ים קטנים הקרובים לחסם תחתון תיאורטי על גודל UHS אפשרי. אנחנו מציגים תוצאות עבור ערכים שונים של k ו-L, ועל ידי החלתם על גנומים אמיתיים מראים ש-UHS אכן משפר על פני מינימייזרים. בפרט, DOCKS משתמש בפחות מ-40% מה-10-יות הנדרשות ע"י השיטה הטובה ביותר שהיתה ידועה קודם לכסות את הגנום האנושי.

התוכנה והקבוצות UHS שייצרנו זמינים ב github.com/Shamir-Lab/DOCKS וב- acgt.cs.tau.ac.il בהתאמה. העבודה פורסמה בתור:

Orenstein, Y.*, Pellow, D.*, Marcais, G., Shamir, R., & Kingsford, C. (2017). **Designing small universal *k*-mer hitting sets for improved analysis of high-throughput sequencing.** *PLoS Computational Biology, 13*(10), e1005777.

בפרק 3 מוצגת כלי לסיווג רצפים מטאגנומיים (metagenomic) הבאים ממקור פלסמידי (plasmid). פלסמידים הם מולקולות DNA מעגליים המופיעים בתאי חיידקים שאינם חלק מהגנום של החיידק. הם יכולים להכיל גנים המקודדים לפונקציות מטאבוליות כמו עמידות לאנטיביוטיקה ולמנגנונים לשכפול והעברת עותקים שלהם לתוך חיידקים אחרים בסביבה. דרך אחת לגלות פלסמידים חדשים וללמוד על הדינמיקה שלהם היא ריצוף מטאגנומי, שמרצף ביחד את כל התוכן הגנומי מדגימה סביבתית. ההרכבה של הקריאות הקצרות מריצוף של דגימה כזו מייצרת עירבוב של רצפים ארוכים יותר הנקראים קונטיגים (contigs), שמקורם במכל המינים שנמצאים בדגימה. אלגוריתם מסווג (classifier) משתמש בתכונות של הרצף כדי לזהות איזה קונטיגים הגיעו במקור מפלסמידים.

בעבודה זו יצרנו מסווג מבוסס רגרסיה לוגיסטית בשם PlasClass שמסווג קונטיגים של פלסמידים על סמך שכיחות ה k-יות שלהם. PlasClass הצליח לסווג רצפים בדיוק טוב יותר. ספציפית, בדקנו את טיב הסיווג של PlasClass על סימולציות, על נתונים פומביים מחיידקים בודדים, ועל דגימות פלסמידום ופלסמידים שהורכבו מדגימות מטאגנומיות. הביצועים של PlasClass היו טובים משל כלים לסיווג רצפים מטאגנומים מתקדמים אחרים, במיוחד על רצפים קצרים יותר המהווים את רוב הקונטיגים בהרכבה מטאגנומית, מה שמאפשר לו להשיג ציוני F1 גבוהים יותר בסיווג רצפים ממגוון רחב של מערכי נתונים. PlasClass גם משתמש בפחות זמן וזיכרון באופן משמעותי. ניתן להשתמש ב-PlasClass כדי לסווג בקלות פלסמיד ורצפי גנום חיידקיים בדגימות מטאגנומיות או של חיידקים בודדים.

הכלי זמין מ: github.com/Shamir-Lab/PlasClass. העבודה פורסמה בתור:

Pellow, D., Mizrahi, I., & Shamir, R. (2020). **PlasClass improves plasmid sequence classification.** *PLoS Computational Biology, 16*(4), e100778

בפרק 4 אני מציג כלי להרכבת רצפים של פלסמידים בדגימות מטאגנומיות, בשם SCAPP. SCAPP פועל על גרף ההרכבה (assembly graph) של דגימה מטאגנומית. האלגוריתם מוצא בצורה יעילה מסלולים מעגליים בגרף שהם מכוסים בעומק אחיד על ידי קריאות בדגימה. המסלולים מכילים קונטיגים שיש להם סבירות גבוהה להגיע מפלסמיד ונמנע מלכלול קונטיגים שמקורם בגנום חיידקי בסבירות גבוהה. בסוף, האלגוריתם גם בודק אם הרצף מכיל גנים שהם ייחודיים לפלסמידים כדי לשלול מסלולים שאינם פלסמידים אמיתיים. השווינו את SCAPP לכלים אחרים להרכבת פלסמידים והראינו שיפורים גם בסימולציות וגם על נתונים אמיתיים. כדי להעריך את הביצועים של הכלים האלו על נתונים אמיתיים, פיתחנו שיטת השוואה חדשה. השותפים שלנו במעבדה של פרופ' איציק מזרחי בבן גוריון ריצפו אותה דגימה פעמיים - פעם כריצוף מטאגנומי רגיל, ופעם אחרי הרצת פרוטוקול שפיתחו להעשרת ריכוז הד.נ.א. הפלסמידי. השוואת הפלסמידים שהכלים מרכיבים מהדגימה

תקציר

רקע כללי

הניתוח של נתוני ריצוף ד.נ.א. בתפוקה גבוהה (high throughput sequencing) איפשר האצה של גילויים ביולוגים [50, 114, 13, 71]. בשל הנפחים ההולכים וגדלים של נתונים מריצוף בתפוקה גבוהה, ניתוחים חישוביים הפכו למאתגרים יותר וחייבים להיות יעילים יותר בצריכת זיכרון ובזמן ריצה. טכנולוגיות חדשות, והניתוחים החדשים שהם מאפשרים, הופכים את האתגר למורכב יותר.

הצורך לנתח נתוני ריצוף הולכים וגדלים חייב התקדמות אלגוריתמית תאורטית רצופה במשך עשרים השנים האחרונות. כמה התקדמויות בולטות בחזית זו הינם: מערך הסיומת [67] (suffix array) והתמרת בורווס-וילר [55] (Burrows Wheeler Transform) למיפתוח (indexing) ועימוד רצפים, BLAST לעימוד לוקאלי [1], פילטר בלום [11] (Bloom filter) לשאילתות הסתברותיות לגבי הכלה של k-יות [113], גרף דה ברוין (de Bruijn graph) להרכבת רצפים [91], סקיצות של רצפים על ידי מינימייזרים [98] (minimizers), ו-MinHash עבור עימוד מקורב של רצפים [84], ועוד רבים אחרים. כלים חדשים לניתוח רצפים נשענים על מסד אלגוריתמי מורכב עוד יותר.

בנוסף לפיתוח תיאורטי של אלגוריתמים ומבני נתונים, ההתקדמות בניתוח נתונים מניסויי ריצוף דרשה הנדסת תוכנה טובה, היוריסטיקות אפקטיביות ו-"טריקים" יישומיים שסוחטים יותר ויותר יעילות חישובית משיטות אלו. ביולוגיה מולקולרית, עם יותר חריגים מאשר כללים [51], מסבכת את העניינים עוד יותר. בנוסף להתקדמויות האלגוריתמיות, כלים יעילים חייבים לנצל ידע ביולוגי קודם ,להישען על מודלים ביולוגים, לפתח היוריסטיקות אד-הוקיות, ולטפל במקרים מיוחדים.

בתיזה זו אני מציג את עבודתיי בשתי החזיתות – התקדמות תיאורטית בסיסית המשפרת את היעילות החישובית והביצועים של כלים לניתוח נתוני ריצוף, ופיתוח ומימוש של כלים ביואינפורמטיים יעילים להרכבת פלסמידים, המשתמשים בידע ביולוגי לשיפור התוצאות שלהם. במקרה הראשון, עבדנו על בעיית קבוצת פגיעה אוניברסלית (Universal Hitting Set, UHS), ופיתחנו אלגוריתם בשם DOCKS, שמוצא UHS קומפקטית בצורה היוריסטית. השתמשנו ב-UHS שייצר האלגוריתם כדי ליצור שיטה לבחירת k-מרים עם צפיפות נמוכה יותר מאשר שיטות אחרות המשתמשות במינימייזרים רגילים. במקרה השני, פיתחתי שיטות לסיווג רצפים של פלסמידים ולהרכבת פלסמידים על בסיס נתונים מדגימות מטאגנומיות. פיתחתי את המסווג PlasClass ואת מרכיב הפלסמידים SCAPP, המשתמשים בידע ביולוגי על פלסמידים כדי להיות מדויקים יותר.

תוצאות

בפרק 2 אני מציג את הבעיה של מציאת קבוצת פגיעה אוניברסלית קטנה (UHS Problem). בעיה זו מוגדרת על ידי פרמטרים L ו-k כאשר $k < L$, ומבקשת למצוא תת-קבוצה קטנה $U_{k,L}$ של כל ה k-יות (רצפים באורך k) כך שכל רצף באורך L"נפגע" על ידי איבר מ $U_{k,L}$, כלומר אחת מה- k-יות מופיעה בו. לבעיה זו אין כיום פתרון יעיל לערכים של k ו L הרלבנטיים לריצוף. בעבודה זה פיתחנו שיטה היוריסטית יעילה שמייצרת קבוצות UHS קטנות יחסית בהינתן k ו L. השיטה מתרגמת את הבעיה לבעיית כיסוי מסלולים בגרף דה ברוין מלא ועובדת בשני שלבים: ראשית, היא מחפשת קבוצת k-יות מינימלית שהורדתה מגרף הדה ברוין המלא הופכת אותו לחסר מעגלים. קיים אלגוריתם ידוע ומאד יעיל למטרה זו. שנית, האלגוריתם מחשב בצורה איטרטיבית את מספר המסלולים באורך L-k העוברים בכל צומת הנשאר בגרף על ידי תכנות דינאמי. בכל איטרציה, מורידים את הצומת עם הכי הרבה מסלולים שעוברים דרכו עד שלא קיים אף מסלול באורךL-k . קבוצת הצמתים שהאלגוריתם הוריד

תמצית

נפח הנתונים המיוצרים על ידי ריצוף ד.נ.א. גדל בקצב מסחרר: טכנולוגיות חדשות, כגון ריצוף עם קריאות ארוכות וריצוף תאים בודדים, וכן סוגי נתונים חדשים, כמו למשל ריצוף מטאגנומי, נכנסים לשימוש פעם אחר פעם. כתוצאה מכך, דרוש שיפור מתמיד של השלבים הבסיסיים בעיבוד נתוני ריצוף, כגון הרכבה ומיפוי של רצפים, כדי להתמודד עם סוגי הדאטה החדשים תוך שמירה על זמן ריצה סביר ועל יעילות השימוש בזיכרון. בתיזה זו אני מציג את תרומותיי לאתגר זה, בשלושה כיוונים. ראשית, הצגנו את הרעיון של קבוצת פגיעה אוניברסלית (Universal Hitting Set, UHS), קבוצה קומפקטית של k-יות (תת-רצפים באורך k) שניתן להשתמש בהן כדי לייצג ביעילות רצפים ארוכים יותר. אפשר להשתמש ב UHS כדי להגדיר שיטת בחירת k-יות עם צפיפות נמוכה שיכולה לשפר את יעילות הזיכרון במגוון רחב של פעולות עיבוד רצפים בסיסיות. פיתחנו אלגוריתם למציאת UHS קומפקטית והראינו את יעילותו המעשית. שנית, פיתחנו שיטות כדי לאפשר הרכבה (assembly) של רצפי פלסמידים שלמים מניסויים מטאגנומים. זה דרש מאיתנו להתמודד עם בעיה שלישית: פיתוח של מסווג רצפים המזהה פלסמידים טוב יותר, המשלב ידע ביולוגי בצורה יעילה כדי לשלול הרכבות שגויות. הכלים שפיתחנו מאפשרים למיקרוביולוגים לחקור את התוכן הגנטי, מבנה האוכלוסייה, והאקולוגיה של פלסמידים מתוך נתונים מדגימות מטאגנומיות.

אוניברסיטת תל-אביב TEL AVIV UNIVERSITY

הפקולטה למדעים מדויקים ע"ש ריימונד ובברלי סאקלר

בית הספר למדעי המחשב ע"ש בלבטניק

שיטות לניתוח יעיל של נתונים מניסויי ריצוף גדולים

חיבור לשם קבלת תואר "דוקטור לפילוסופיה"

מאת דוד פלו

בהנחייתו של פרופ' רון שמיר

הוגש לסנאט של אוניברסיטת ת"א

ספטמבר 2022