A Mixed Integer Linear Programming Algorithm for Plasmid Binning

Aniket Mane(✉), Mahsa Faizrahnemoon, and Cedric Chauve

Department of Mathematics, Simon Fraser University, Burnaby, BC, Canada
{amane,mfaizrah,cedric.chauve}@sfu.ca

Abstract. The problem of analysing bacterial isolates in order to detect plasmids has been widely studied. With the development of Whole Genome Sequencing (WGS) technologies, several approaches have been proposed to bin contigs into putative plasmids. Reference-based approaches aim to bin contigs by mapping or comparing their sequences against databases of previously identified plasmids or plasmid genes. On the other hand, de novo approaches use contig features such as read coverage and length for plasmid binning. Hybrid approaches that combine both strategies have also been proposed recently.

We present PlasBin a mixed integer linear programming based hybrid approach for plasmid binning. We evaluate the performance of several binning methods on a real data set of bacterial samples.

1 Introduction

Antimicrobial resistance (AMR) has recently emerged as a major public health concern. AMR is developed in bacteria through the acquisition of antimicrobial genes. This is often facilitated by mobile genetic elements (MGEs). Mobile genetic elements enabling antimicrobial resistance can be exchanged between bacteria via, among other mechanisms, evolutionary events known as horizontal gene transfer (HGT) [8]. The family of plasmids falls under the umbrella of MGEs. Plasmids are circular, double-stranded segments of DNA that are separate from chromosomes and are known to carry AMR genes. Thus, the identification and detection of plasmids from whole-genome sequencing (WGS) data is important for understanding the propagation of AMR genes.

With the advent of DNA sequencing technologies, it is now possible to obtain WGS data at a relatively low cost. For bacterial genomes, reads sequenced using WGS technologies are generally processed by a genome assembler, such as Unicycler [19] or SPAdes [4], to produce an assembly, often in the form of contigs; recent assemblers such as SPAdes and Unicycler also generate an assembly graph. The problem of detecting plasmids from the output of an assembler can be studied at three different levels: classification, binning and assembly.

At the *classification* level, the aim is to classify whether a contig originates from a plasmid or a chromosome. Erstwhile tools such PlasmidFinder [9] map query contigs against a reference database of plasmid markers. Several approaches

L. Jin and D. Durand (Eds.): RECOMB-CG 2022, LNBI 13234, pp. 279–292, 2022.
https://doi.org/10.1007/978-3-031-06220-9_16

based on machine learning have also been utilized for contig classification. These approaches attempt to learn the genomic signal, often in the form of k-mer composition, from a training data set of known plasmid sequences. Current state-of-the-art tools include mlplasmids [3], PlasClass [16], PlasFlow [11] and RFPlasmid [6].

At the *binning* level, the aim is to group contigs into bins, with the expectation that the contigs in a bin likely originate from the same plasmid. Various approaches have been proposed for plasmid binning. MOB-recon – part of the recently developed MOB-suite [17] – is a reference-based method that maps contigs against a database of known plasmid sequences and a database of plasmid marker sequences known as replicons and relaxases. Plasmids from the reference database are clustered according to their sequence similarity. Contigs are then binned together if they match with plasmids belonging to the same reference cluster. Putative plasmid bins thus constructed are then filtered using the presence/absence of known replicons or relaxases. The reliance on a reference database can potentially hinder the ability of reference-based methods to identify novel plasmids. De-novo approaches do not depend on reference sequences. Instead, they use contig features as well as, for some methods, additional information from the assembly graph. Plasmids often occur in the genome in several copies. Thus, the coverage of plasmids is expected to be significantly different than that of chromosomes. Recycler [18] peels off cycles from the assembly graph assuming uniform coverage of sequenced plasmids and using length thresholds. PlasmidSPAdes [1] also relies on coverage features: it estimates the chromosomal coverage from the assembly graph, removes contigs with similar coverage as that of the chromosome and then computes putative plasmid bins from the connected components of the remaining graph. However, this strategy may fail to identify plasmids that have a low copy number. Lastly some plasmid binning methods employ a hybrid strategy by combining ideas from both of the above approaches. Such methods first use a reference databases to assign weights to the contigs. They then use contig features such as coverage and topography of the assembly graph to separate contigs into putative plasmid bins. Gplas [2] uses the signal from known plasmidic sequences to assign a probability that the contig is plasmidic in origin; gplas uses mlplasmids [3] to compute this probability. Contigs classified as chromosomal are removed from the assembly and the remaining contigs are then separated into plasmid bins, using a graph-theoretical method. HyAsP [14] is another hybrid plasmid binning tool. It first computes the plasmid genes density of each contig by mapping the contigs against a database of known plasmid genes. It then uses a greedy heuristic to extract cycles or paths from the WGS assembly graph as putative plasmid bins, with the goal to maximize the plasmid gene density over the selected contigs while maintaining a uniform read coverage.

Finally, at the *assembly* level, contigs that are in the same plasmid bin are linearly or circularly ordered. Note that plasmid assembly methods by default are also useful for contig classification and plasmid binning. From the plasmid binning approaches mentioned above, Recycler and HyAsP have actually been designed as assembly methods.

In the present work, we focus on the plasmid binning problem, motivated by the fact that, for downstream analysis, the most useful information is in the form of groups of genes that belong to a given plasmid, more than in the order of these genes along the plasmid sequence; for example, plasmid typing can be done from the gene content and does not consider gene order [10,12]. Hence, although our method does perform plasmid assembly, we focus on plasmid binning and evaluate the results accordingly. We propose a hybrid approach that uses a mixed integer linear programming (MILP) formulation for plasmid binning. We compare our method, PlasBin, against HyAsP, plasmidSPAdes, MOB-suite and gplas on a data set of 133 bacterial samples and evaluate their performance using precision, recall and F1 statistics. Our experiments indicate that the hybrid methods generally perform better than other approaches, and that our method PlasBin obtains the best average F1 score, slightly outperforming HyAsP.

2 Hybrid Approach for Plasmid Binning Using Mixed Integer Linear Programming

We present PlasBin, an optimization based hybrid approach for the plasmid binning problem. PlasBin is based on the same principles as HyAsP, which utilizes a greedy approach to extract paths or cycles from an assembly graph. The evaluation of HyAsP demonstrated that, at the time of its design, it outperformed existing methods. The rationale of PlasBin is that by relying on a global optimization approach, using an objective function based on the HyAsP greedy path extension score, it will be able to avoid the pitfall of locally optimal solutions, while improving some of HyAsP limitations.

2.1 Input: Contigs and the Assembly Graph

The two main input for PlasBin are the contigs and the assembly graph, obtained using Unicycler [19][1]. Every contig has two extremeties, a head and a tail, denoted by c_h and c_t for a contig c. Pairs of contigs that are potentially adjacent in the genome sequence are connected to each other via edges of the assembly graph. Although, we do not consider the order of contigs for the purpose of plasmid binning, the edges help in ensuring that the contigs chosen to belong to a plasmid bin form a path or a cycle in the graph. We represent the edges of the assembly graph as pairs of contig extremeties: an edge between c_h and d_h is represented as (c_h, d_h). PlasBin also takes as input a database of genes known to occur in plasmids, called as the reference database from now.

Contig Features. PlasBin considers several features associated to a contig, namely its %GC content, sequencing coverage, length and plasmid gene density.

[1] We rely on Unicycler as it is a widely used bacterial genome assembler, but any assembler providing an assembly graph can be used.

- The %GC content of a sequence is the proportion of the sequence consisting of either G or C bases. It is a useful feature as the %GC content of plasmids is generally expected to be lower than that of chromosomes [15]. For contig c, we denote by gc_c its %GC content.
- Plasmids might occur in multiple copies in a cell, as opposed to the chromosome, which is single-copy. Thus, plasmidic contigs are likely to exhibit higher sequencing coverage, while contigs originating from the same plasmid are expected to have similar coverage. We define the coverage of a contig as the normalized coverage computed by Unicycler, *i.e.* the average base coverage of the contig normalized by the median coverage of the contigs in the assembly graph; thus chromosomal contigs are expected to have a normalized coverage close to 1. The coverage of contig c is denoted by rc_c.
- We denote by ℓ_c the length of contig c.
- Taking a cue from HyAsP, we associate each contig with its *plasmid gene density*, which is the proportion of the sequence matching with a set of genes known to occur in plasmids. We compute the plasmid gene density of a contig by mapping genes from the reference database to contigs using blastn (version 2.6.0) [7], considering matches that show an identity of at least 95% and cover at least 95% of the gene. The plasmid gene density of contig c is denoted by gd_c.

Seed Contigs. Based on the features described above, some contigs are more likely to be part of a plasmid than others. We refer to these contigs as seeds, a notion introduced in HyAsP. Seeds are contigs that pass certain thresholds on the features length, coverage and gene density. The parameters defining seed contigs can be modified by the user, but the default values are taken from HyAsP. The seed eligibility of contig c, a boolean feature, is denoted by s_c.

Preprocessing: Duplicating High-Coverage Contigs. PlasBin assumes that a plasmid has been sequenced with near-uniform coverage; thus all contigs from a given plasmids, provided they are not repeated elsewhere in the genome. In order to handle the possibility of a contig to be repeated within a plasmid bin, or in different plasmid bins, we duplicate contigs based on their coverage as follows: if a contig c has normalized coverage k we create $\lceil k \rceil$ nodes $c_1, c_2, ..., c_{\lceil k \rceil}$ and any edge incident to c in the original graph is duplicated into edges incident to every c_i in the modified graph. The other attributes (gene density, GC content, seed eligibility and length) remain the same as for the original contig c. As a result of this preprocessing, every node of the modified assembly graph has normalized coverage 1; this is a major difference between PlasBin and HyAsP, where the Unicycler normalized coverage was used as is in the greedy path extension heuristic.

2.2 PlasBin Workflow

PlasBin works iteratively. During each iteration an MILP computes a path in the preprocessed assembly graph that optimizes an objective function defined

in terms of contig features; by the way we preprocessed the original assembly graph, this path can be a walk in the original assembly graph, thus allowing for repeated contigs. Contigs on this path represent a plasmid bin. At the end of each iteration, we obtain as output both the list of contigs belonging to a plasmid bin and a list of edges defining in the corresponding path. The MILP also enforces that at least one seed belongs to the path.

Once a plasmid bin is generated by the MILP, we update the assembly graph by removing contigs from the bin as well as the edges incident to these contigs. PlasBin then moves on to the next iteration with the updated assembly graph as input. This process continues until no seed contig is present.

Next, similarly to HyAsP, we classify every plasmid bin as putative or questionable based on certain thresholds on the overall plasmid gene density and the cumulative length of the contigs in the bin.

2.3 MILP Formulation

We now describe the MILP used in every iteration to compute a path in the provided assembly graph. This is the central part of PlasBin, aimed at replacing the greedy heuristic of HyAsP.

Decision Variables: To each contig, contig extremity and edge in the assembly graph, we associate a decision binary variables to indicate if it is part of the computed optimal path. For each contig, we also associate decision variables pertaining to %GC content, gene density and seed eligibility. These variables take the value of the respective feature if and only if the contig in question is part of the solution ($x_c = 1$), otherwise they take value 0. Finally, we introduce a continuous variable MGC that records the mean %GC content of contigs in the computed path. For a graph $AG = (V, E)$, the total number of variables is $O(|E|)$. Table 1 below describes the decision variables. We discuss later in this section how we handle variables whose formulation is a priori non-linear.

Table 1. Decision variables used in the MILP; p refers to the computed path.

Variable	Type	Description
x_c	Binary	$x_c = 1$ if node $c \in p$
$x_{c,ext}$	Binary	$x_{c,ext} = 1$ if extremity ext of contig $c \in p$
y_e	Binary	$y_e = 1$ if edge $e \in p$
GC_c	Continuous	$GC_c = gc_c$ if $c \in p$, 0 otherwise
S_c	Boolean	$S_c = s_c$ if $c \in p$, False otherwise
L_c	Integer	$L_c = \ell_c$ if $c \in p$, 0 otherwise
L_p	Integer	$L_p = \sum_{c \in p} \ell_c$
MGC	Continuous	$\sum_{c \in p}(gc_c * \ell_c)/L_p$

Objective Function: We formulate the MILP as a minimization problem, that aims to find a path in the assembly graph that minimizes a linear combination of the (negated) plasmid gene density and of a term measuring the deviation of %GC content across the contigs forming the path.

- Gene density: The aim is to maximize the plasmid gene density over the whole path. The gene density of a contig is a number between 0 and 1 so, to account for the high variability of contigs length, we weight the gene density of selected contigs by their lengths and integrate the term $-\sum_c (gd_c)*L_c$ in the objective function (the sum is over all contigs).
- %GC content deviation: We aim to minimize the sum of the deviations between the %GC content of selected contigs and the mean %GC content of the whole path. This feature was shown in HyAsP to contribute greatly to the method accuracy. Similarly to the gene density, we weight the %GC of each contig in the path by its length and incorporate the term $\sum_c (|MGC - gc_c|)*L_c$ in the objective function. Here, MGC is the mean %GC content of the path introduced in Table 1.

The objective function is thus, a linear combination of the two terms above, with each term being weighted equally. Note that the optimality criteria for HyAsP also contains a term pertaining to coverage uniformity. However, we circumvent the need for computing the deviation in contig coverage through the modifications made to the assembly graph as described in Sect. 2.1.

An important remark is that, if we were to exclude the %GC content term from the objective function, the problem to address would be a shortest path problem, with the additional constraint that the optimal path should contain a seed. This would be amenable to efficient shortest paths algorithms. However, as mentioned above, the %GC content term is important toward selecting paths that are more likely to originate from plasmids, which makes the optimization problem we address more complicated.

Constraints: We provide now an overview of the constraints required to solve the problem of finding a path in the assembly graph that is optimal with regard to the objective function.

1. Constraint to ensure the presence of a seed in the path: This is done by requiring that the sum of the binary decision variable x_c over all seed contigs is at least 1.
2. Constraints to handle variables in the denominator: In order to minimize %GC content deviation, we also need to compute the mean %GC content of the path, MGC. The mean %GC content of a plasmid is given by:
   $MGC = \frac{\sum_c gc_c * \ell_c * x_c}{L_p}$

   This is not a linear constraint as L_p is a variable in the denominator. However, it is possible to obtain a linear relaxation for the above constraint through the use of McCormick envelopes [13]. This is possible since x_c is a binary variable and thus, has definite upper and lower bounds.

3. Constraints to handle absolute values: The objective function contains the absolute value of the difference between the mean %GC of the path (MGC) and individual %GC (GC_c). This in itself is not linear. So, we introduce a difference variable d_c which is the absolute value of the difference between the mean %GC and contig c %GC for a specific path. For each contig we add the following constraints: $d_c >= MGC - GC_c$, $d_c >= GC_c - MGC$.
4. Constraints to define a path: In every iteration, we want the MILP to output a path. This is the most challenging aspect of our formulation, as it is notorious that modelling the search of an arbitrary path in a graph through a polynomial number of linear constraints is not possible. To handle this issue, we start from degree constraints that enforce that every contig extremity belongs to at most one selected edge. However, this leads to the possibility that the generated solution consists of disjoint components, all but one of which are cycles. Thus, we would like to add constraints to eliminate potential cycles. To address this issue, we use the delayed constraint generation method [5]: whenever the solution found by the MILP contains cycles, we add constraints explicitly forbidding these cycles and repeat. This results into an iterative process that goes on until we obtain a singular component defining a path as our solution. A possible scenario in this approach is an exponential number of iterations. To avoid this, we bound the number of iterations.

The number of constraints, excluding cycles-forbidding constraints added in the delayed constraint generation step, is in $O(|E|)$.

Implementation. The PlasBin MILP is solved using Gurobi solver (version 9.1.2). The code was written in python, using the Gurobi API. Each individual iteration of the MILP was allowed to run for a maximum of 4 h until the optimality gap threshold of 5% is reached. If the time threshold was reached before the optimality gap threshold, the MILP output the feasible solution at the time of stopping. Any solution thus obtained would consist of a path and possibly some cycles. Constraints to prevent these cycles were then added for the next iteration. The code is available from https://github.com/cchauve/PlasBin.

3 Experimental Results

We evaluated the results of the PlasBin on a dataset of 133 bacterial genomes and 377 plasmids from a collection of real plasmids compiled in [17]. This follows the experimental evaluation of HyAsP. Some of the methods such as HyAsP, MOB-suite and PlasBin are partly or entirely reference-based. To simulate the use of a realistic reference database, we split our data into a reference set and a test set. Samples released before 19 December 2015 were used to build the reference database and those released after that date formed the test set. The test set consisted of 66 samples, with a total of 147 plasmids, for which Illumina sequencing data was re-assembled using Unicycler to provide contigs and assembly graphs (we refer to [17]). We evaluated the following methods: PlasBin, HyAsP, MOB-recon, plasmidSPAdes and gplas.

3.1 Performance Comparison of Plasmid Binning Tools

The results of PlasBin were evaluated using precision, recall and F1-score. Putative plasmid bins (PlasBin, HyAsP) or plasmid contigs (MOB-suite, plasmidSPAdes, gplas) were mapped against the reference plasmids database using BLAST+ [7]. The precision (resp. recall) was defined as the proportion of predicted plasmid lengths matched with the reference plasmids (resp. the proportion of reference plasmid lengths covered by alignments to the predicted plasmids). Although PlasBin provides questionable plasmid bins in its output, they are not to be considered as part of the predicted bins. Hence, we focus on the statistics for putative plasmid bins.

Gplas requires the use of mlplasmids to compute the probability of contigs to originate from a plasmid. Since mlplasmids currently supports only 4 species (*Escherichia coli, Enterococcus faecium, Klebsiella pneumoniae and Acinetobacter baumannii*), gplas was evaluated only over the subset of samples belonging to these species.

The performance of PlasBin was first compared against HyAsP, MOB-suite and plasmidSPAdes for all the 66 test samples. The mean statistics for all the tools are given in Table 2, while full statistics are shown in Fig 1.

Table 2. Mean statistics for various plasmid assembly tools

Tool	Precision	Recall	F1
PlasBin	0.859	0.884	0.860
HyAsP	0.873	0.855	0.849
plasmidSPAdes	0.743	0.821	0.723
MOB-suite	0.869	0.644	0.686

We observe that the hybrid methods HyAsP and PlasBin have comparable performances. HyAsP and MOB-suite show the best precision, attaining a precision of over 0.9 for many samples, while PlasBin consistently yields a precision close to 0.8. On the other hand, plasmidSPAdes yields lower precision than the other methods. In terms of recall, we observe that plasmidSPAdes, PlasBin and HyAsP can account for most of the plasmid contigs with a recall of 0.8 or more. PlasBin performed the best, consistently identifying 80% of true plasmid contigs, with HyAsP and plasmidSPADes only slightly lagging behind PlasBin. MOB-suite on the other hand showed weaker recall than all the other tools.

Overall, considering the F1-score, we observe that PlasBin and HyAsP generally perform better than the other tools, with a slight advantage for PlasBin. Both methods consistently have an F1 score over 0.8. The performances of plasmidSPAdes and MOB-suite varied across the samples. MOB-suite sometimes outperformed PlasBin in terms of precision but not in terms of recall. On the other hand, plasmidSPAdes showed slightly poorer precision and recall than the hybrid methods. This suggests that the two-pronged strategy of using reference-based information as well as known plasmid characteristics tends to yield better results.

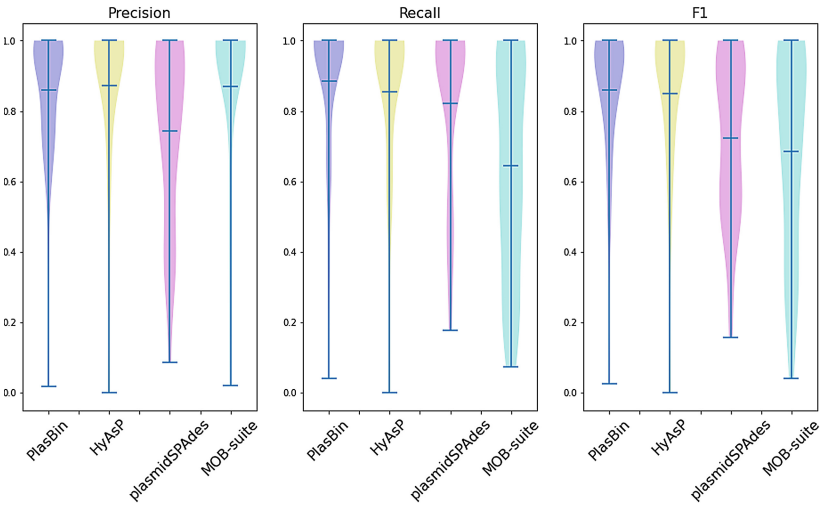

Fig. 1. Precision, recall and F1-score statistics for all tools (except gplas)

We then used gplas on the subset of samples belonging to *Escherichia coli, Enterococcus faecium or Klebsiella pneumoniae*[2] (Table 3 and Fig. 2).

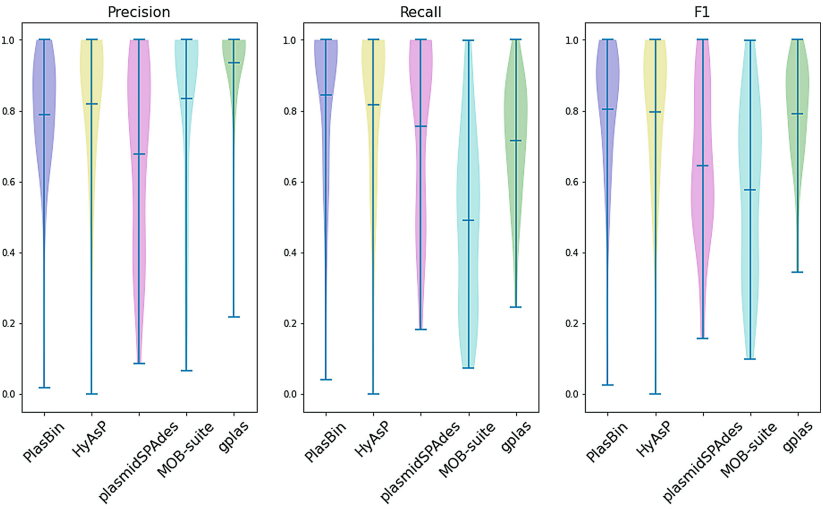

Fig. 2. Precision, recall and F1-score statistics for all tools for three species supported by gplas

[2] Our data set did not contain any samples from *Acinetobacter baumannii*.

Table 3. Mean statistics for species supported by gplas

Tool	Precision	Recall	F1
PlasBin	0.788	0.843	0.804
HyAsP	0.820	0.817	0.807
plasmidSPAdes	0.623	0.784	0.632
MOB-suite	0.835	0.491	0.576
gplas	0.935	0.714	0.791

We observe that gplas tends to avoid false positives and has the best precision, which is expected given it relies on species-specific training datasets. For the three species supported by gplas, PlasBin had median precision over 0.8, Overall, in terms of F1-score, PlasBin and HyAsP performed better than the other three methods, with gplas slightly behind. An interesting comment is the fact that the hybrid methods HyAsP and PlasBin compared well to gplas despite this tool using species-specific training data.

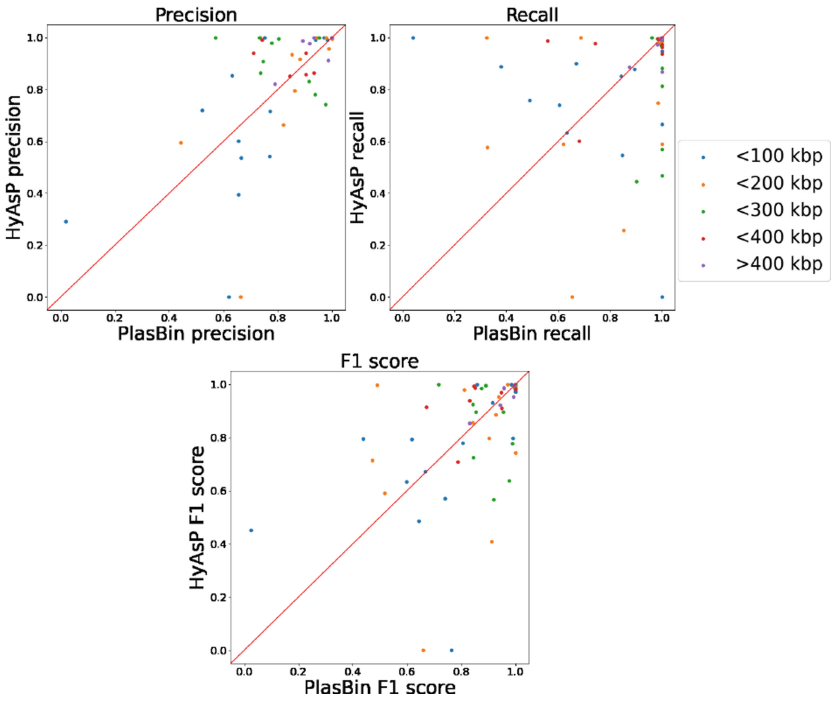

Fig. 3. Precision, recall and F1-score for PlasBin and HyAsP. Each point in the scatter-plot represents a sample. Points have been assigned a color according to the cumulative length of true plasmids in the sample.

3.2 Comparison of PlasBin and HyAsP

Motivated by the fact that we expected PlasBin, by design, to outperform HyAsP, which we did not observe to a significant point, we compared the results of PlasBin and HyAsP for each sample, with a special focus on the total length of the true and predicted plasmids. The scatterplots of the precision, recall and F1-score per sample of both tools are shown alongside in Fig 3.

The precision and recall were computed while accounting for the length of the plasmids in the set of predicted and reference plasmids respectively. From Fig 4, it can be seen that in most cases, the sum of the lengths of incorrectly predicted contigs is small, often less than 5000 bp and consistently less than 20000 bp. However, considering the size of plasmids in the data set, a relatively small absolute error often translates to a large relative error thereby resulting in low precision.

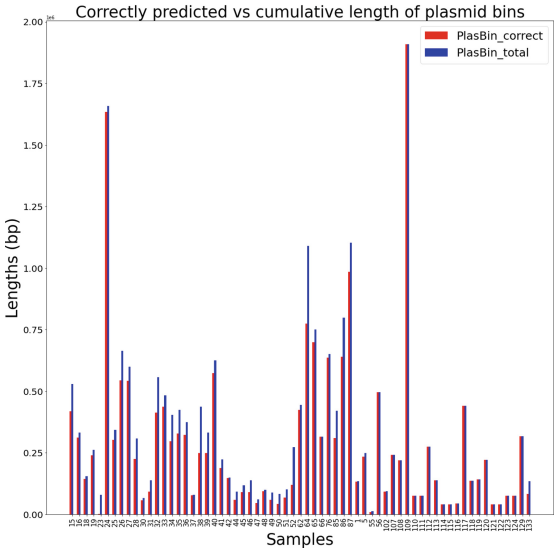

Fig. 4. Cumulative length of correctly predicted contigs (red) and of all the contigs in bins predicted by PlasBin (blue), for each sample (Color figure online)

3.3 Computational Footprint

PlasBin relies on an MILP, that can require large computing resources (time and memory), while HyAsP, being a greedy heuristic, induces a low computational footprint. We ran our experiments on a standard laptop computer, with a quad-core processor and 16 GB of memory. On average, over all considered samples, PlasBin required a little more than an hour of computation to complete (maximum of 8 h), with a memory footprint close to 6 GB (maximum 10 GB). So, while being more costly than HyAsP, PlasBin has a reasonable computational footprint. The details have provided in Table 4.

Table 4. Runtime statistics for various tools (in minutes)

Tool	Min	Max	Median	Mean
PlasBin	5	473	22	117
HyAsP	1	19	3	4
plasmidSPAdes	4	363	16	43
MOB-suite	3	15	5	8
gplas	2	12	3	4

4 Discussion

In this paper, we presented a hybrid method for plasmid binning using, based on replacing the greedy heuristic of HyAsP by an exact MILP approach. The performance of PlasBin was compared against other state-of-the-art plasmid binning methods, namely, HyAsP, MOB-suite, plasmidSPAdes and gplas. In some instances, gplas and MOB-suite showed better precision, however, PlasBin as well as HyAsP consistently showed better recall and a better F1-score, with a slight advantage to PlasBin in terms of the F1-score. As a result, these two methods had the best overall performance of all the five methods used, lending support to the hybrid approach.

Despite having an objective function similar to the extension criteria used in HyAsP, the two methods function in a different manner. At any point in the HyAsP algorithm, there are a limited number of alternatives to extend the plasmid chain being assembled. As a result, HyAsP can only search locally. PlasBin on the other hand uses a branch-and-bound algorithm and performs the search for the optimal solution globally. In many instances, particularly with regards to precision, the greedy heuristic performs better than PlasBin. For recall, however, PlasBin consistently performs better than the other approaches. These observations suggests that the MILP might be lenient in its choice of plasmid contigs. The presence of chromosomal contigs in plasmids predicted by PlasBin may be explained by the following: the MILP tries to maximize the gene density within a plasmid bin, which may result in a plasmid prediction with two contigs with high plasmid gene density joined by a chain of low gene density contigs. If contigs from this intermediate chain indeed originate from chromosomes, it might result in poor precision.

The objective function of PlasBin is a linear combination of two terms, one each pertaining to the gene density and the %GC content deviation of selected contigs. Currently, we consider each term to be equally important. However, it may be worthwhile to explore different linear combinations of the two terms.

Gplas consistently showed a high level of precision. This can likely be credited to the the novel strategy of identifying the plasmid contigs before binning. Once the plasmid contigs are identified using mlplasmids, gplas only considers those contigs in the next step of separating contigs into plasmid bins. This allows gplas

to avoid false positives and also significantly reduces the size of the graph needed to be traversed.

Our MILP formulation, especially its objective function, is amenable to incorporate other features of the contigs. The comparison with gplas suggests that incorporating the probability of a contig to originate from a plasmid could be beneficial, without the need to reduce drastically the assembly graph to the subgraph of likely plasmidic contigs as is done by gplas. While gplas is a species-specific tool for contigs classification, that requires a species-specific training datasets, there are other plasmid contig classification tools such as PlasClass [16] that are more general. Integrating contigs classification results and lenient graph pruning is a promising research avenue.

References

1. Antipov, D., Hartwick, N., Shen, M.W., Raiko, M., Lapidus, A.L., Pevzner, P.A.: plasmidSPAdes: assembling plasmids from whole genome sequencing data. Bioinformatics **32**(22), 3380–3387 (2016). https://doi.org/10.1093/bioinformatics/btw493
2. Arredondo-Alonso, S., et al.: gplas: a comprehensive tool for plasmid analysis using short-read graphs. Bioinformatics **36**(12), 3874–3876 (2020). https://doi.org/10.1093/bioinformatics/btaa233
3. Arredondo-Alonso, S., et al.: mlplasmids: a user-friendly tool to predict plasmid- and chromosome-derived sequences for single species. Microb. Genom. **4**(11), e000224 (2018). https://doi.org/10.1099/mgen.0.000224
4. Bankevich, A., et al.: Spades: a new genome assembly algorithm and its applications to single-cell sequencing. J. Comput. Biol. **19**(5), 455–477 (2012). https://doi.org/10.1089/cmb.2012.0021
5. Bertsimas, D., Tsitsiklis, J.: Introduction to Linear Optimization. Athena Scientific, 1st edn. (1997)
6. van der Graaf-van Bloois, L., Wagenaar, J.A., Zomer, A.L.: RFPlasmid: predicting plasmid sequences from short-read assembly data using machine learning. Microb. Genom. **7**(11) (2021). https://doi.org/10.1099/mgen.0.000683
7. Camacho, C., et al.: BLAST+: architecture and applications. BMC Bioinf. **10**, 421 (2009). https://doi.org/10.1186/1471-2105-10-421
8. Carattoli, A.: Plasmids and the spread of resistance. Int. J. Med. Microbiol. **303**(6), 298–304 (2013). https://doi.org/10.1016/j.ijmm.2013.02.001
9. Carattoli, A., et al.: In silico detection and typing of plasmids using plasmidfinder and plasmid multilocus sequence typing. Antimicrob. Agents Chemother. **58**(7), 3895–3903 (2014). https://doi.org/10.1128/AAC.02412-14
10. Dewar, A., et al.: Plasmids do not consistently stabilize cooperation across bacteria but may promote broad pathogen host-range. Nat. Ecol. Evol. **5**(12), 1624–1636 (2021). https://doi.org/10.1038/s41559-021-01573-2
11. Krawczyk, P., Lipinski, L., Dziembowski, A.: Plasflow: predicting plasmid sequences in metagenomic data using genome signatures. Nucleic Acids Res. **46**(6), e35 (2018). https://doi.org/10.1093/nar/gkx1321
12. Luo, L., et al.: Comparative genomics of Chinese and international isolates of Escherichia albertii: population structure and evolution of virulence and antimicrobial resistance. Microb. Genom. **7**(12) (2021). https://doi.org/10.1099/mgen.0.000710

13. McCormick, G.P.: Computability of global solutions to factorable nonconvex programs: Part I—convex underestimating problems. Math. Program. **10**(1), 147–175, e000224 (1976). https://doi.org/10.1007/BF01580665
14. Müller, R., Chauve, C.: HyAsP, a greedy tool for plasmids identification. Bioinformatics **35**(21), 4436–4439 (2019). https://doi.org/10.1093/bioinformatics/btz413
15. Nishida, H.: Comparative analyses of base compositions, DNA sizes, and dinucleotide frequency profiles in archaeal and bacterial chromosomes and plasmids. Int. J. Evol. Biol. **2012**, 342482 (2012). https://doi.org/10.1155/2012/342482
16. Pellow, D., Mizrahi, I., Shamir, R.: Plasclass improves plasmid sequence classification. PLoS Comput. Biol. **16**(4), 1–9 (2020). https://doi.org/10.1371/journal.pcbi.1007781
17. Robertson, J., Nash, J.: MOB-suite: software tools for clustering, reconstruction and typing of plasmids from draft assemblies. Microb. Genom. **4**(8), e000206 (2018). https://doi.org/10.1099/mgen.0.000206
18. Rozov, R., et al.: Recycler: an algorithm for detecting plasmids from de novo assembly graphs. Bioinformatics **33**(4), 475–482 (2016). https://doi.org/10.1093/bioinformatics/btw651
19. Wick, R.R., Judd, L.M., Gorrie, C.L., Holt, K.E.: Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads. PLoS Comput. Biol. **13**(6), 1–22 (2017). https://doi.org/10.1371/journal.pcbi.1005595