GraphBin2: Refined and Overlapped Binning of Metagenomic Contigs Using Assembly Graphs

Vijini G. Mallawaarachchi
Research School of Computer Science, College of Engineering and Computer Science, Australian National University, Canberra, Australia
vijini.mallawaarachchi@anu.edu.au

Anuradha S. Wickramarachchi
Research School of Computer Science, College of Engineering and Computer Science, Australian National University, Canberra, Australia
anuradha.wickramarachchi@anu.edu.au

Yu Lin
Research School of Computer Science, College of Engineering and Computer Science, Australian National University, Canberra, Australia
yu.lin@anu.edu.au

Abstract

Metagenomic sequencing allows us to study structure, diversity and ecology in microbial communities without the necessity of obtaining pure cultures. In many metagenomics studies, the reads obtained from metagenomics sequencing are first assembled into longer contigs and these contigs are then binned into clusters of contigs where contigs in a cluster are expected to come from the same species. As different species may share common sequences in their genomes, one assembled contig may belong to multiple species. However, existing tools for contig binning only support non-overlapped binning, *i.e.*, each contig is assigned to at most one bin (species). In this paper, we introduce GraphBin2 which refines the binning results obtained from existing tools and, more importantly, is able to assign contigs to multiple bins. GraphBin2 uses the connectivity and coverage information from assembly graphs to adjust existing binning results on contigs and to infer contigs shared by multiple species. Experimental results on both simulated and real datasets demonstrate that GraphBin2 not only improves binning results of existing tools but also supports to assign contigs to multiple bins.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Computational genomics

Keywords and phrases Metagenomics binning, contigs, assembly graphs, overlapped binning

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.8

Supplementary Material Our program can be found at `https://github.com/Vini2/GraphBin2`.

Acknowledgements We would like to thank the anonymous reviewers for their valuable comments. Furthermore, this research was undertaken with the assistance of resources and services from the National Computational Infrastructure (NCI Australia), an NCRIS enabled capability supported by the Australian Government.

1 Introduction

With the advent of high throughput sequencing approaches, the field of metagenomics has enabled us to access and study the genetic material of entire microbial communities [25, 32]. A microbial community is usually a complex mixture of multiple species and recovering these species is crucial to understand the behaviour and functions within such communities. To characterise the composition of a sample, we cluster metagenomic sequences into groups that represent different taxonomic groups such as species, genera or higher levels [28]. This process

is known as *metagenomics binning*. Although it is possible to bin reads directly (before assembly) [1, 8, 10, 18, 23, 27, 33], reads are usually too short to enable accurate binning results [34]. Hence, a typical approach in metagenomics analysis starts from assembling short reads into longer contigs and then bin these resulting contigs into groups representing different taxonomic groups [28].

Existing contig-binning tools can be divided into two categories, (1) reference-based and (2) reference-free. Reference-based binning approaches [3, 15, 20, 37] rely on a database of reference genomes and thus may not be applicable in many metagenomic samples when reference genomes are not available. Reference-free binning tools use unsupervised approaches to group contigs into unlabelled bins which correspond to different taxonomic groups solely based on the information obtained from the contigs [28]. These reference-free binning methods become very useful when analysing environmental samples where many species are not found in the current reference databases [16]. Most of the reference-free tools make use of the composition and/or abundance (coverage) information of contigs to bin them [2, 12, 13, 14, 31, 36, 38]. Although contigs are assembled from reads using assembly graphs, most existing binning tools do not use the information of the assembly graph. More recently, GraphBin [19] has been developed to use the connectivity information in the assembly graph to refine the binning results of existing tools because contigs connected to each other in the assembly graph are more likely to belong to the same taxonomic group [5].

Different bacterial genomes in a metagenomic sample may share similar genes and genomic regions [26], which is a major challenge in assembling metagenomic reads into contigs [22]. Therefore, some assembled contigs from metagenomic reads may be shared by multiple species in the sample. However, very few contig-binning tools support overlapped binning (*i.e.*, assigning shared contigs to multiple species). S-GSOM [7] abstracts the flanking sequences of highly conserved 16S rRNA and incorporates them into Growing Self-Organising Maps (GSOM) to bin contigs into overlapping bins. MetaPhase [6] uses Hi-C reads to scaffold assembled contigs into assemblies of individual species and allows certain contigs to belong to multiple species. However, the applications of S-GSOM and MetaPhase are limited due to their required additional sequencing effort (*e.g.*, 16S RNA or Hi-C sequencing). As shared contigs correspond to shared vertices between different genomic paths on the assembly graph [22], it is worth investigating whether it is possible to infer such shared contigs from the assembly graph without additional sequencing requirements.

In this paper, we present GraphBin2, the new generation of GraphBin, to refine binning results using the assembly graph. While GraphBin only uses the topology information of the assembly graph, GraphBin2 improves the algorithms to adjust existing binning results and to support overlapped binning based on both the connectivity and coverage information of assembly graphs. Experimental results show that GraphBin2 not only improves existing binning results, but also infers contigs that may belong to multiple species.

2 Methods

Figure 1 denotes the workflow of GraphBin2. The preprocessing steps of GraphBin2 assemble reads into contigs using the assembly graph and then bin the contigs (*i.e.*, assign coloured labels to contigs) using existing contig-binning tools. GraphBin2 takes this labelled assembly graph as the input, removes unsupported labels, corrects the labels of inconsistent vertices, propagates labels to unlabelled vertices and finally infers vertices with multiple labels (colours).

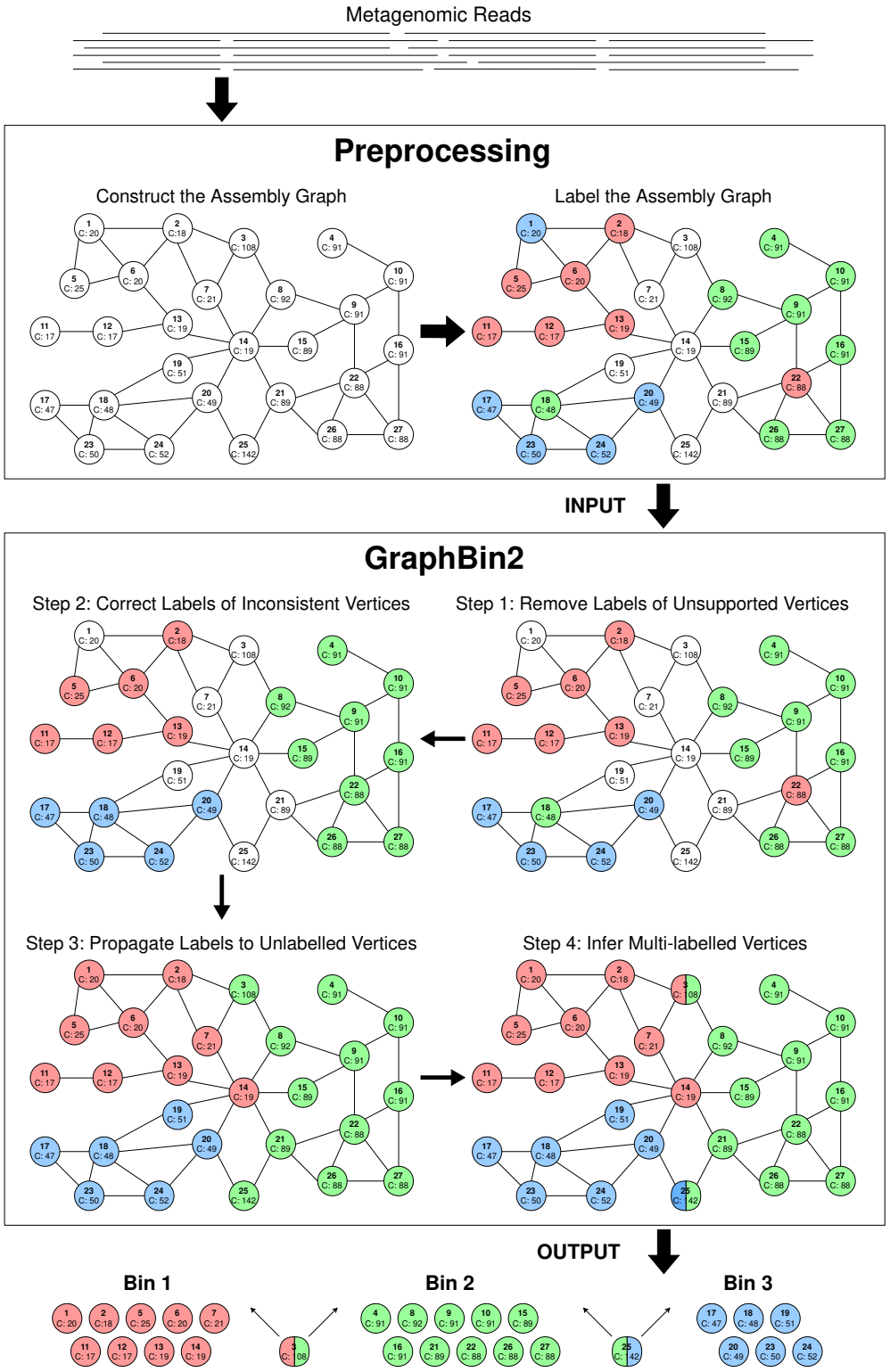

Figure 1 The workflow of GraphBin2.

2.1 Preprocessing

In this step, we assemble the next generation reads (*e.g.*, Illumina reads with length ranging from 75bp to 300bp) into contigs using the assembly graph. There are two dominant paradigms for genome assembly: overlap-layout-consensus (or string graphs) [21] and de Bruijn graphs [24]. We select two representative assemblers from each paradigm, SGA [30] and metaSPAdes [22] respectively, to demonstrate the adaptability of GraphBin2. In the assembly graph, each vertex represents a contig with *coverage* denoting the average number of reads that map to each base of the contig and each edge indicates a significant overlap between a pair of contigs. In an ideal case, a genome corresponds to a path in the assembly graph and its genomic sequence corresponds to the concatenation of contigs along this path. Hence, if two contigs are connected by an edge in the assembly graph, they are more likely to belong to the same genome. Previous studies [5, 19] have shown that the connectivity information between contigs can be used to refine binning results. In the assembly graph of metagenomic datasets, different genomes usually correspond to different paths in the assembly graph. If two genomes share a common contig (*e.g.*, unresolved "interspecies repeat" [22]), the corresponding vertex would be shared by two genomic paths in the assembly graph.

After assembling reads into contigs using assembly graphs, GraphBin2 uses an existing contig-binning tool to derive an initial binning result. Note that most of the existing tools for binning contigs require a minimum length for the contigs (*e.g.,* 1,000bp for MaxBin2 [38] and SolidBin [36], 500bp for BusyBee Web [16] and 1500bp for MetaBAT2 [13]). Therefore, many short contigs in the assembly graph will be discarded, resulting in low recall values as a common limitation of existing binning tools. For example, 65% of the contigs in the metaSPAdes assembly of the **Sharon-All** dataset were discarded by MaxBin2 due to their short length.

2.2 Step 1: Remove Labels of Unsupported Vertices

A linear (or circular) chromosome usually corresponds to a path (or a cycle) that traverses multiple vertices in the assembly graph. If two contigs belong to the same chromosome, they are likely to be connected by a path which consists of other contigs from the same chromosome. Therefore, a labelled vertex is defined as *supported* if and only if one of the following conditions hold.

- It is an isolated vertex
- It directly connects to a vertex of the same label
- It connects to a vertex of the same label through a path that consists of only unlabelled vertices

Otherwise, a labelled vertex is defined as *unsupported*. Note that the definition of *unsupported* vertices in GraphBin2 is more strict than *ambiguous* vertices in GraphBin.[1] For example, in the initial labelled assembly graph of Figure 1, vertex 2 in red is supported by vertex 6 in red as they are directly connected. Note that vertex 18 in green is also supported by vertex 15 in green as there exists a path (*i.e.*, $18 \rightarrow 19 \rightarrow 14 \rightarrow 15$) between them that traverses only unlabelled vertices (*i.e.*, 19 and 14). However, vertex 1 in blue is unsupported as it cannot reach another blue vertex through a path consisting of only unlabelled (white coloured) vertices.

[1] In GraphBin, a vertex i is denoted as an *ambiguous* vertex if at least one of its closest labelled vertices has a label that is different than the label of the vertex i.
An *ambiguous* vertex in GraphBin may be *supported* (in GraphBin2) by another vertex of the same label if they are directly connected or connected through a path consisting of only unlabelled vertices.
An *unsupported* vertex in GraphBin2 is always *ambiguous* in GraphBin.

To check whether a labelled vertex is *supported* or *unsupported*, a naive approach is to perform a breadth-first-search from each labelled vertex. A refined algorithm first initialises all labelled vertices as unsupported and scans the graph to identify all labelled vertices that are either isolated or directly connected to a vertex of the same label and classifies them as supported vertices. This refined algorithm then uses breadth-first-search to find all connected components that consist of only unlabelled vertices and for each component $Component$ stores a set of labelled vertices $N(Component)$ that are connected to vertices in $Component$. If multiple labelled vertices in $N(Component)$ have the same label, these vertices are supported because they connect to each other through a path that consists of only unlabelled vertices in $Component$. GraphBin2 removes the labels for all unsupported vertices because these labels may not be reliable. For example, the label of the unsupported vertex 1 is removed by GraphBin2 in Step 1 of Figure 1.

2.3 Step 2: Correct Labels of Inconsistent Vertices

After Step 1, each non-isolated labelled vertex v is supported by at least one vertex with the same label. The closer two vertices are in the assembly graph, the more likely they have the same label. For each vertex v, we introduce a *labelled score*, $S(v, x)$, for each label x by considering all vertices of label x that are directly connected to v or connected to v through a path that consists of only unlabelled vertices. A vertex t of label x contributes to $S(v, x)$ by $2^{-D(v,t)}$ where $D(v, t)$ is the shortest distance between v and t using only unlabelled vertices. This distance is measured by the number of edges in a path and $D(v, t) = 1$ if v and t are directly connected. Therefore, the *labelled score* $S(v, x)$ is the sum of contributions from all vertices of label x that are directly connected to v or connected to v through a path that consists of only unlabelled vertices. In Step 1 of Figure 1, vertex 17 contributes 1/2 to $S(18, blue)$ because $D(17, 18) = 1$ and vertex 8 contributes 1/8 to $S(18, green)$ because $D(8, 17) = 3$. The labelled score of $S(18, blue)$ is 2 to which all four blue vertices 17, 20, 23 and 24 contribute 1/2 respectively while $S(18, green) = 5/16$ to which vertex 8 contributes 1/8, vertex 15 contributes 1/8 and vertex 26 contributes 1/16.

A labelled vertex v of label x is defined as *inconsistent* if and only if the *labelled score* of its current label x times α is less than or equal to the *labelled score* of another label y where α is a parameter, *i.e.*, $\alpha \times S(v, x) \leqslant S(v, y)$. We have set $\alpha = 1.5$ in the default settings of GraphBin2. In Step 1 of Figure 1, vertex 18 in green is an inconsistent vertex because $1.5 \times S(18, green) = 1.5 \times 5/16 = 0.47$ is less than $S(18, blue) = 2$.

Again, GraphBin2 uses the breadth-first-search to check if a labelled vertex is *inconsistent*. GraphBin2 corrects the label of an inconsistent vertex v to another label that maximises the labelled score. For example, GraphBin2 corrects the label of vertex 18 from green to blue and corrects the label of vertex 22 from red to green (refer from Step 1 to Step 2 in Figure 1).

2.4 Step 3: Propagate Labels to Unlabelled Vertices

As existing contig-binning tools discard contigs due to their short lengths in the initial binning, many vertices are still unlabelled in the current assembly graph. In this step, we will propagate existing labels to the remaining unlabelled vertices using the assembly graph. There are two intuitions behind this label propagation process. Firstly, vertices that are closer to each other in the assembly graph are more likely to have the same label. Secondly, vertices with similar coverages are more likely to have the same label because contigs from the same genome usually have similar coverages [39, 12]. GraphBin2 uses both the connectivity and coverage information of the assembly graph to propagate the labels.

For each unlabelled vertex v with coverage $c(v)$ (*i.e.*, coverage of the contig that corresponds to the vertex), a candidate propagation action $(D(v,t), |c(v)-c(t)|, t, v)$ is recorded as a tuple where t is the nearest labelled vertex to v, $c(t)$ is the coverage of t and $D(v,t)$ is the shortest distance between v and t (as defined in Step 2). Given two candidate propagation actions, (d_1, c_1, t_1, v_1) and (d_2, c_2, t_2, v_2), GraphBin2 will execute (d_1, c_1, t_1, v_1) before (d_2, c_2, t_2, v_2), *i.e.,* propagating the label of t_1 to v_1 before propagating the label of t_2 to v_2, if $(d_1 < d_2)$ or $(c_1 < c_2$ and $d_1 = d_2)$. In other words, GraphBin2 puts more emphasis on the connectivity information than the coverage information because the edges in the assembly graph are expected to be more reliable than the coverage information on vertices, especially for vertices corresponding to short contigs (which are discarded by initial binning tools).

GraphBin2 first uses the breadth-first-search to compute all candidate propagation actions for unlabelled vertices and sort them into a ranked list according to the order defined above. At each iteration, GraphBin2 executes the first candidate propagation action and then updates the ranked list of candidate propagation actions. Note that one unlabelled vertex receives its label at each iteration and updating the ranked list of candidate propagation actions can be done efficiently by breadth-first-search from this unlabelled vertex. Please refer to the Supplementary Material Section A to see a step-by-step label propagation process from Step 2 to Step 3 in Figure 1. Note that this label propagation process in GraphBin2 improves on the label propagation algorithm in GraphBin by incorporating both the connectivity and coverage information in the assembly graph. So far, GraphBin2 does not generate multi-labelled vertices. In the next step, we will show how GraphBin2 uses the labelling, connectivity and coverage information together on the assembly graph to infer multi-labelled vertices.

2.5 Step 4: Infer Multi-Labelled Vertices

Contigs belonging to multiple genomes correspond to multi-labelled vertices in the assembly graph. What are the characteristics of shared contigs between multiple species? Firstly, a contig shared by multiple genomes may connect other contigs in these genomes. Secondly, the coverage of a contig shared by multiple genomes should be equal to the sum of coverages of these genomes in the ideal case. After label propagation, vertices of the same label are likely to form connected components in the assembly graph and multi-labelled vertices are likely to be located along the borders between multiple connected components where distinct labels meet and have a coverage similar to the sum of the average coverages of multiple components that they belong to.

GraphBin2 checks labelled vertices that are connected to vertices of multiple different labels. The average coverage of a connected component P is calculated by $\frac{\sum c(i) \times L(i)}{\sum L(i)}$ for each vertex i in the connected component P, where $c(i)$ is the coverage of the vertex i and $L(i)$ is the length of the contig corresponding to vertex i. Assume v is a labelled vertex v from a component P, the coverage of v is $c(v)$ and the average coverage of P is $c(P)$. When $c(v)$ is larger than $c(P)$ and v is connected to other components $P_1, P_2, \ldots, P_k$ with different labels, it is possible that v also belongs to one or more components (in addition to P). For example, if v belongs to P, P_i and P_j in the ground-truth, the coverage of v, $c(v)$, is expected to be close to the sum of average coverages of the above three components, $c(P) + c(P_i) + c(P_j)$. In fact, finding which components in $\{P_1, P_2, \ldots, P_k\}$ that v also belongs to (in addition to P) can be modelled as the following subset sum problem [9]. Given a set of positive numbers $\{c(P_1), c(P_2), \ldots, c(P_k)\}$, find a subset whose sum is or is closest to $c(v) - c(P)$. Then v will be assigned to the corresponding components in this subset as well as to P. Note that it is possible that the selected subset is empty and thus v only belongs to P.

In all of our experiments, the maximum number of different components that a vertex connects to in the assembly graph is less than 5. We use a brute-force way to enumerate all possible combinations of components and find out the combinations that best explain the observed coverages. For example, after Step 3 in Figure 1, vertex 3 in green connects to another red component. The coverage of vertex 3 is 108 while the average coverage of the green component is 95 and the average coverage of the red components is 19. Because the coverage of vertex 3 (108) is closer to the sum of average coverages of green and red components (95+19=114) compared to the average coverage of the green component (95), vertex 3 is assigned both green and red labels. Similarly, the coverage of vertex 25 (142) is closer to the sum of average coverages of green and blue components (95+49=144) compared to the average coverage of the green component (95). Hence, vertex 25 is assigned both green and blue labels. In the same assembly graph after Step 3 in Figure 1, vertex 14 in red does not gain any other labels because its own coverage is closest to the average coverage of the red component (19) compared to other possible combinations (*i.e.*, red+blue, red+green, green+blue and red+green+blue).

3 Experimental Setup

3.1 Datasets

3.1.1 Simulated Datasets

We simulated three metagenomic datasets according to the species found in the simMC+ dataset [38]. Three datasets were simulated each containing 5 species (referred as **Sim-5G**), 10 species (referred as **Sim-10G**) and 20 species (referred as **Sim-20G**) respectively. Paired-end reads were simulated using the tool InSilicoSeq [11] modelling a MiSeq instrument with 300bp mean read length. More details about the simulated datasets can be found in Table 1 and Supplementary Material Section B.

3.1.2 Real Datasets

We used the preborn infant gut metagenome, commonly known as the **Sharon** dataset [29] (NCBI accession number *SRA052203*). There are 18 Illumina (Illumina HiSeq 2000) runs available for this dataset. One run *SRR492184* is included as a representative dataset (referred as **Sharon-1**) and all the 18 Illumina runs are combined to form the **Sharon-All** dataset in our experiments. Further details can be found in Supplementary Material Section B.

3.2 Tools Used

To derive the assembly graph, there are two dominant assembly paradigms, de Bruijn graphs [24] and overlap-overlap-layout-consensus (or string graphs) [21]. We selected one representative tool from each paradigm to show the effectiveness of GraphBin2. To represent the de Bruijn graph paradigm, we used metaSPAdes[22] (from SPAdes version 3.13.0 [4]) with its default parameters to generate the assembly graph. As for the overlap-layout-consensus paradigm, we selected SGA (version 0.10.15) [30] to derive the assembly graph.

We used MaxBin2 (version 2.2.5) [38] with default parameters and SolidBin (version 1.3) [36] in `SolidBin-SFSmode` to obtain the initial binning results for our experiments. MaxBin2 and SolidBin are considered as hybrid contig-binning tools as they use both the composition and coverage information. They make use of tetranucleotide frequencies and

Table 1 Details about the simulated datasets.

Dataset	Species present	Genome size	Coverage	Abundance
Sim-5G	Acetobacter pasteurianus	2.9 Mb	115×	28%
	Aeromonas veronii	4.6 Mb	72×	28%
	Amycolatopsis mediterranei	10.4 Mb	26×	22%
	Arthrobacter arilaitensis	3.9 Mb	41×	13%
	Azorhizobium caulinodans	5.4 Mb	20×	9%
Sim-10G	Acetobacter pasteurianus	2.9 Mb	357×	25%
	Aeromonas veronii	4.6 Mb	225×	25%
	Amycolatopsis mediterranei	10.4 Mb	80×	20%
	Arthrobacter arilaitensis	3.9 Mb	128×	12%
	Azorhizobium caulinodans	5.4 Mb	62×	8%
	Bacillus cereus	5.3 Mb	58×	7%
	Bdellovibrio bacteriovorus	3.8 Mb	11×	1%
	Bifidobacterium adolescentis	2.1 Mb	20×	1%
	Brachyspira intermedia	3.4 Mb	11×	1%
	Campylobacter jejuni	1.7 Mb	21×	1%
Sim-20G	Acetobacter pasteurianus	2.9 Mb	705×	23%
	Aeromonas veronii	4.6 Mb	445×	23%
	Amycolatopsis mediterranei	10.4 Mb	157×	18%
	Arthrobacter arilaitensis	3.9 Mb	253×	11%
	Azorhizobium caulinodans	5.4 Mb	123×	7%
	Bacillus cereus	5.3 Mb	114×	7%
	Bdellovibrio bacteriovorus	3.8 Mb	22×	1%
	Bifidobacterium adolescentis	2.1 Mb	40×	1%
	Brachyspira intermedia	3.4 Mb	21×	1%
	Campylobacter jejuni	1.7 Mb	41×	1%
	Candidatus Pelagibacter ubique	1.3 Mb	54×	1%
	Chlamydia trachomatis	1.1 Mb	64×	1%
	Clostridium acetobutylicum	4.0 Mb	18×	1%
	Corynebacterium diphtheriae	2.5 Mb	28×	1%
	Cyanobacterium UCYN	1.5 Mb	47×	1%
	Desulfovibrio vulgaris	3.6 Mb	20×	1%
	Ehrlichia ruminantium	1.5 Mb	47×	1%
	Enterococcus faecium	3.0 Mb	24×	1%
	Erysipelothrix rhusiopathiae	1.8 Mb	39×	1%
	Escherichia coli	5.0 Mb	14×	1%

coverages of reads with different machine learning approaches to bin contigs. Note that both MaxBin2 and SolidBin only bin contigs which are longer than 1,000bp by default. We also compared GraphBin2 with its predecessor GraphBin [19]. The commands used to run all the assembly and binning tools can be found in Supplementary Material Section C.

3.3 Evaluation Criteria

Since the reference genomes of the simulated datasets were known, we used BWA-MEM [17] to align the contigs to their reference genomes to determine the ground truth species to which the contigs actually belonged to. If at least 50% of a contig aligns to a species, then a contig is considered to belong to this species. Note that a contig may be considered to belong to multiple species if multiple such alignments exist. To reduce the effect of random alignments between short contigs and multiple genomes, a contig is considered to belong to multiple species when its length is at least 1,000bp long. Furthermore, isolated contigs (corresponding vertices with zero degree in the assembly graph) were not considered for the ground-truth set of the datasets.

For the Sharon dataset, we considered the annotated contigs from 12 species which are available at `https://ggkbase.berkeley.edu/carrol/organisms` as references. A process similar to the simulated datasets were followed for the Sharon datasets to determine the origin species of contigs and contigs belonging to multiple species.

To evaluate the binning results of MaxBin2 [38], SolidBin [36], GraphBin [19] and GraphBin2, we used the metrics (1) precision, (2) recall and (3) F1-score which have been used in previous studies [2, 19, 35]. The binning result is denoted as a $K \times S$ matrix where K is the number of bins identified by the binning tool and S is the number of species available in the ground truth. In this matrix, the element a_{ks} denotes the number of contigs binned to the k^{th} bin and belongs to the s^{th} species. Note that contigs belonging to multiple species are not included in this matrix. $Unclassified$ denotes the number of contigs that are unclassified or discarded by the tool. Following are the definitions and equations that were used to calculate the precision, recall and F1-score.

$$Precision = \frac{\sum_k max_s\{a_{ks}\}}{\sum_k \sum_s a_{ks}} \tag{1}$$

$$Recall = \frac{\sum_s max_k\{a_{ks}\}}{(\sum_k \sum_s a_{ks} + Unclassified)} \tag{2}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

To evaluate the detection of multi-labelled vertices corresponding to contigs that may belong to multiple species, we used the criteria (1) sensitivity (also known as true positive rate or recall) which measures the proportion of actual positives that are correctly identified, (2) specificity (also known as true negative rate) which measures the proportion of actual negatives that are correctly identified and (3) balanced accuracy as follows.

$$Sensitivity = \frac{TP}{TP + FN} \tag{4}$$

$$Specificity = \frac{TN}{TN + FP} \tag{5}$$

$$Balanced\ accuracy = \frac{Sensitivity + Specificity}{2} \tag{6}$$

Here TP refers to the true positives (*i.e.*, the number of multi-labelled vertices correctly assigned with multiple labels), FP refers to the false positives (*i.e.*, the number of single-labelled vertices incorrectly assigned with multiple labels), FN refers to the false negatives (*i.e.*, the number of multi-labelled vertices incorrectly assigned with a single label) and TN refers to the true negatives (*i.e.*, the number of single labelled vertices correctly assigned with a single label). We use the balanced accuracy because the dataset is imbalanced; *i.e.*, the number of multi-labelled contigs is much smaller than the number of single-labelled contigs.

4 Results and Discussion

4.1 Binning Results

Table 2 Comparison of binning results of MaxBin2 [38], GraphBin [19] and GraphBin2 (on top of MaxBin2 results) using assembly graphs built by metaSPAdes [22]. The best values are highlighted in bold.

Dataset	No. of bins identified	Evaluation Criteria	**MaxBin2**	**GraphBin** with **MaxBin2** results	**GraphBin2** with **MaxBin2** results
Sim-5G	5	Precision	92.28%	**99.80%**	99.03%
		Recall	44.16%	97.08%	**99.03%**
		F1 score	59.74%	98.42%	**99.03%**
Sim-10G	10	Precision	90.24%	99.77%	**99.78%**
		Recall	38.21%	98.66%	**99.78%**
		F1 score	53.69%	99.21%	**99.78%**
Sim-20G	21	Precision	89.48%	**98.28%**	97.78%
		Recall	41.37%	94.06%	**97.71%**
		F1 score	56.59%	96.12%	**97.74%**
Sharon-1	5	Precision	75.46%	89.28%	**90.02%**
		Recall	31.59%	61.44%	**62.53%**
		F1 score	44.54%	72.79%	**73.80%**
Sharon-All	11	Precision	83.80%	90.02%	**90.09%**
		Recall	28.55%	82.04%	**83.25%**
		F1 score	42.58%	85.84%	**86.53%**

Table 3 Comparison of binning results of SolidBin [36], GraphBin [19] and GraphBin2 (on top of SolidBin results) using assembly graphs built by metaSPAdes [22]. The best values are highlighted in bold.

Dataset	No. of bins identified	Evaluation Criteria	**SolidBin**	**GraphBin** with **SolidBin** results	**GraphBin2** with **SolidBin** results
Sim-5G	5	Precision	91.94%	**99.40%**	99.03%
		Recall	44.36%	96.50%	**99.03%**
		F1 score	59.84%	97.93%	**99.03%**
Sim-10G	10	Precision	92.17%	**99.21%**	98.77%
		Recall	39.44%	98.99%	**99.55%**
		F1 score	55.24%	99.10%	**99.16%**
Sim-20G	10	Precision	17.51%	35.30%	**46.05%**
		Recall	8.80%	89.62%	**90.05%**
		F1 score	11.72%	50.65%	**60.94%**
Sharon-1	5	Precision	72.31%	83.98%	**86.93%**
		Recall	30.08%	86.99%	**92.95%**
		F1 score	42.49%	85.46%	**89.84%**
Sharon-All	9	Precision	78.30%	**82.98%**	81.13%
		Recall	22.63%	66.75%	**68.30%**
		F1 score	35.11%	73.99%	**74.16%**

Table 4 Comparison of binning results of MaxBin2 [38], GraphBin [19] and GraphBin2 (on top of MaxBin2 results) using assembly graphs built by SGA [30]. The best values are highlighted in bold.

Dataset	No. of bins identified	Evaluation Criteria	**MaxBin2**	**GraphBin** with **MaxBin2** results	**GraphBin2** with **MaxBin2** results
Sim-5G	5	Precision	93.01%	99.45%	**99.57%**
		Recall	2.70%	99.35%	**99.57%**
		F1 score	5.26%	99.40%	**99.57%**
Sim-10G	9	Precision	97.12%	93.03%	**98.08%**
		Recall	5.21%	89.73%	**94.70%**
		F1 score	9.89%	91.35%	**96.36%**
Sim-20G	20	Precision	**96.30%**	87.66%	94.39%
		Recall	4.03%	85.91%	**92.69%**
		F1 score	7.74%	86.78%	**93.53%**
Sharon-1	5	Precision	91.29%	85.90%	**93.48%**
		Recall	32.90%	76.67%	**80.47%**
		F1 score	48.36%	81.02%	**86.49%**
Sharon-All	8	Precision	63.32%	77.83%	**78.07%**
		Recall	15.85%	37.62%	**39.58%**
		F1 score	25.35%	50.73%	**52.53%**

Table 5 Comparison of binning results of SolidBin [36], GraphBin [19] and GraphBin2 (on top of SolidBin results) using assembly graphs built by SGA [30]. The best values are highlighted in bold.

Dataset	No. of bins identified	Evaluation Criteria	**SolidBin**	**GraphBin** with **SolidBin** results	**GraphBin2** with **SolidBin** results
Sim-5G	5	Precision	93.37%	99.29%	**99.62%**
		Recall	2.71%	99.29%	**99.54%**
		F1 score	5.27%	99.29%	**99.58%**
Sim-10G	9	Precision	85.20%	77.82%	**88.91%**
		Recall	5.05%	75.38%	**93.62%**
		F1 score	9.54%	76.58%	**91.20%**
Sim-20G	19	Precision	**86.25%**	77.07%	83.28%
		Recall	3.86%	64.92%	**77.31%**
		F1 score	7.39%	70.10%	**80.18%**
Sharon-1	4	Precision	94.79%	**97.19%**	96.53%
		Recall	35.78%	90.83%	**91.59%**
		F1 score	51.95%	93.90%	**93.99%**
Sharon-All	5	Precision	60.85%	**76.02%**	75.90%
		Recall	22.33%	47.48%	**47.84%**
		F1 score	32.67%	58.45%	**58.69%**

Table 2 and Table 4 denote the binning results of MaxBin2 [38] and the binning results of GraphBin [19] and GraphBin2 on top of MaxBin2 results for the metaSPAdes [22] assemblies and SGA [30] assemblies, respectively. Table 3 and Table 5 demonstrate the results of SolidBin [36], GraphBin [19] and GraphBin2 on top of SolidBin results for metaSPAdes assemblies and SGA assemblies, respectively. The results in these tables show that GraphBin2 achieves the best performance in most of the scenarios. Both GraphBin and GraphBin2 have shown significant improvements on recall compared to MaxBin2 and SolidBin. While MaxBin2 and SolidBin filter contigs with length shorter than 1,000bp, GraphBin and GraphBin2 are able to bin short contigs using assembly graphs built by either metaSPAdes or SGA. In a few scenarios, GraphBin2 improved on the recall with a bit of a compromise on the precision compared to the GraphBin because GraphBin removes ambiguous labels in the final step. Furthermore, the existence of weak edges (*i.e.*, edges that are not well supported from the data) can form false connections between contigs and can mislead the label propagation process.

4.2 Multi-Labelled Inference Results

One key novelty of GraphBin2 is the introduction of the multiple-labelled inference for contigs. Tables 6, 7, 8 and 9 demonstrate the performance of the GraphBin2 with its multi-labelled inference. It is evident that there is an increase in the number of multi-labelled contigs with the increasing complexity of the dataset.

Table 6 Multi-labelled inference results using GraphBin2 on top of MaxBin2 [38] results for the metaSPAdes assemblies.

Dataset	Ground truth	TP	FP	TN	FN	Sensitivity	Specificity	Balanced accuracy
Sim-5G	2	2	2	512	0	100.00%	99.61%	99.81%
Sim-10G	5	4	3	893	1	80.00%	99.67%	89.83%
Sim-20G	7	4	7	1,393	3	57.14%	99.50%	78.32%
Sharon-1	2	2	1	368	0	100.00%	99.73%	99.86%
Sharon-All	8	4	34	2,692	4	50.00%	98.75%	74.38%

Table 7 Multi-labelled inference results using GraphBin2 on top of SolidBin [36] results for the metaSPAdes assemblies.

Dataset	Ground truth	TP	FP	TN	FN	Sensitivity	Specificity	Balanced accuracy
Sim-5G	2	2	3	511	0	100.00%	99.42%	99.71%
Sim-10G	5	4	3	893	1	80.00%	99.67%	89.83%
Sim-20G	7	4	7	1,393	3	57.14%	99.50%	78.32%
Sharon-1	2	1	1	369	1	50.00%	99.73%	74.86%
Sharon-All	8	1	29	2,700	7	12.50%	98.94%	55.72%

GraphBin2 has assigned correct labels for most of the multi-labelled and single-labelled vertices (*i.e.*, TP+TN). The relatively poor true-positive rate on **Sharon-All** dataset may be due to the poor performance of the initial binning results of MaxBin2 and SolidBin. Moreover, the sequencing noise or contamination in the real metagenomic dataset may also

affect the identification of multi-labelled vertices in the assembly graph. Furthermore, the **Sharon-All** dataset consisted of short reads of 100bp length compared to other datasets and resulted in a very fragmented assembly graph with a large number of nodes and edges.

Table 8 Multi-labelled inference results using GraphBin2 on top of MaxBin2 [38] results for the SGA assemblies.

Dataset	Ground truth	TP	FP	TN	FN	Sensitivity	Specificity	Balanced accuracy
Sim-5G	3	1	5	18,186	2	33.33%	99.97%	66.65%
Sim-10G	2	1	8	32,380	1	50.00%	99.98%	74.99%
Sim-20G	3	1	14	72,776	2	33.33%	99.98%	66.66%
Sharon-1	3	1	1	764	2	33.33%	99.87%	66.60%
Sharon-All	9	1	36	20,905	8	11.11%	99.83%	55.47%

Table 9 Multi-labelled inference results using GraphBin2 on top of SolidBin [36] results for the SGA assemblies.

Dataset	Ground truth	TP	FP	TN	FN	Sensitivity	Specificity	Balanced accuracy
Sim-5G	3	2	6	18,184	1	66.67%	99.97%	83.32%
Sim-10G	2	1	0	32,388	1	50.00%	100.00%	75.00%
Sim-20G	3	1	10	72,780	2	33.33%	99.99%	66.66%
Sharon-1	3	1	0	765	2	33.33%	100.00%	66.67%
Sharon-All	9	2	15	20,925	7	22.22%	99.93%	61.08%

4.3 Visualisation of the Assembly Graph

Figure 2 denotes the labelling of the contigs in the metaSPAdes assembly graph of the **Sim-5G** dataset at different stages as it undergoes the processing of GraphBin2. In Figure 2(a), we can see that some mis-binned contigs are identified (circled in red) as differently coloured contigs within components of a single colour. Figure 2(b) shows the refined assembly graph where GraphBin2 has removed labels of unsupported vertices and corrected labels of inconsistent vertices. After GraphBin2 propagates labels to the remaining unlabelled vertices, the assembly graph will be as denoted in Figure 2(c). Finally, GraphBin2 will detect multi-labelled vertices that correspond to contigs that may belong to multiple species as shown by the black coloured vertices in Figure 2(d).

4.4 Implementation

The source code for the experiments was implemented using Python 3.7.3 and run on a Darwin system with macOS Mojave 10.14.6, 16G memory and Intel Core i7 CPU @ 2.8 GHz with 4 CPU cores. In our experiments, we restrict the depth of the breadth-first-search in Steps 2-3 to be 5 to speed up GraphBin2. Moreover, we have set the parameter $\alpha = 1.5$ by default for GraphBin2. Furthermore, the process of inferring multi-labelled vertices was performed in parallel using multithreading (set to 8 threads by default in GraphBin2).

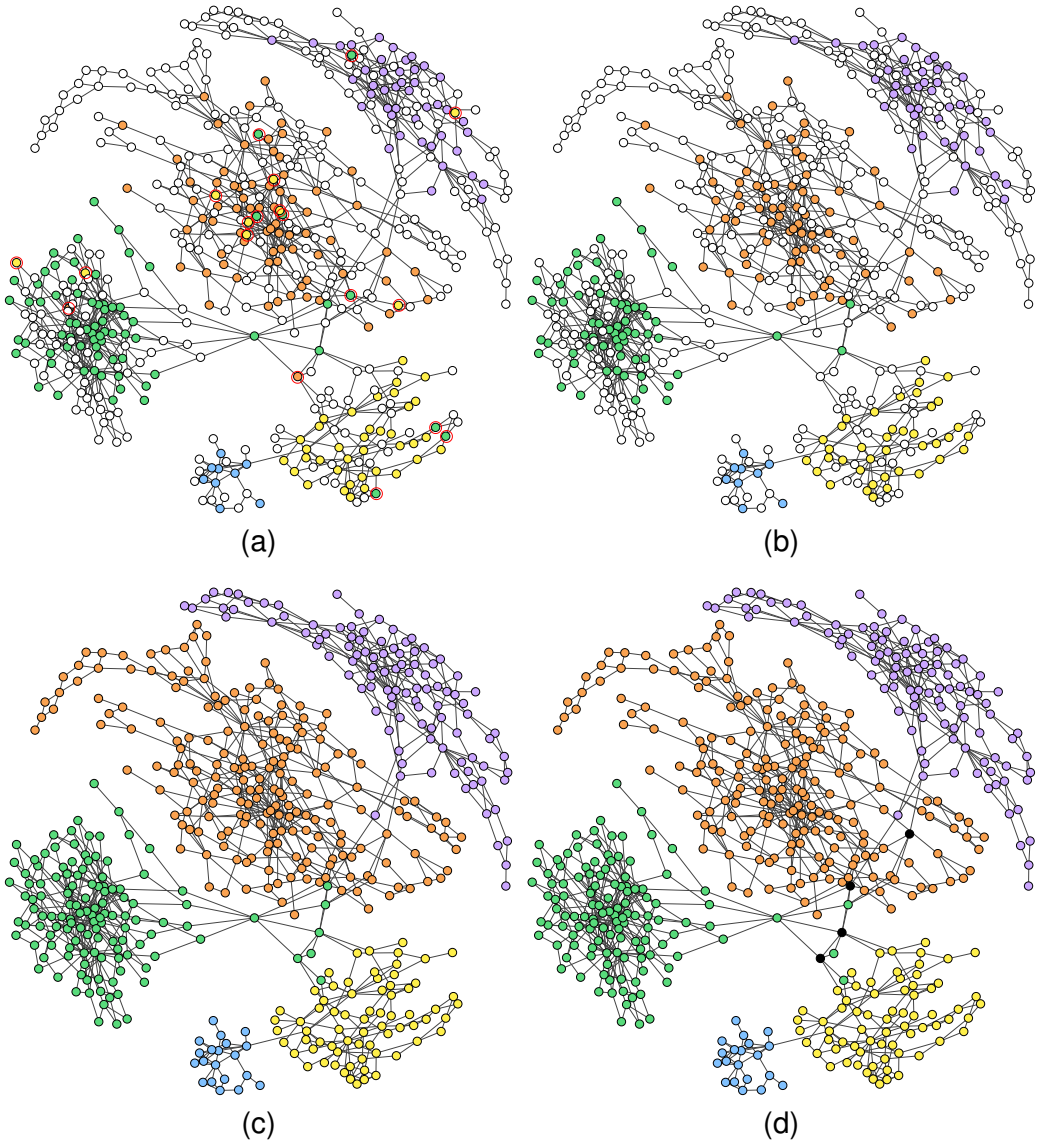

Figure 2 The labelling of the assembly graph of Sim-5G dataset based on (a) the initial MaxBin2 result (mis-binned contigs are circled in red), (b) after removing labels of unsupported vertices and correcting labels of inconsistent vertices, (c) after propagating labels of unlabelled vertices (d) after determining multi-labelled vertices (black coloured vertices) by GraphBin2.

4.5 Running Time and Memory Usage

Table 10 denotes the running times (wall time) and the peak memory used for the **Sharon-1** and **Sharon-All** datasets. MaxBin2 and GraphBin2 executed with 8 threads and SolidBin executed with a single thread. The running times for MaxBin2 and SolidBin only include the times taken to run the main software, excluding the times taken to build the composition and coverage profile files.

GraphBin2 took less than 12 minutes and less than 165 MB of memory to complete executing the **Sharon-All** dataset with 8 threads.

Table 10 Running times (wall time) and peak memory usage for binning using each tool. *s* denotes seconds, *m* denotes minutes and *MB* denotes megabytes.

Dataset	Assembly type	Criteria	MaxBin2	GraphBin2 with MaxBin2 result	SolidBin	GraphBin2 with SolidBin result
Sharon-1	metaSPAdes	Time	9s	5s	6s	5s
		Memory	1,389 MB	45 MB	290 MB	45 MB
	SGA	Time	12s	3s	15s	3s
		Memory	203 MB	33 MB	654 MB	33 MB
Sharon-All	metaSPAdes	Time	30s	10m 50s	2m 7s	11m 12s
		Memory	1,378 MB	163 MB	1,416 MB	163 MB
	SGA	Time	28s	1m 21s	2m 51s	1m 15s
		Memory	241 MB	50 MB	2,612 MB	50 MB

5 Conclusion

In this paper we presented a novel algorithm, GraphBin2, that incorporates the coverage information into the assembly graph as an improvement of GraphBin [19]. While GraphBin uses only the topology of the assembly graph to remove and propagate labels, GraphBin2 makes use of the coverage information on vertices to perform label propagation. Furthermore, GraphBin2 enables the detection of contigs that may belong to multiple species. The performance of GraphBin2 was evaluated against its predecessor and two other binning tools on top of contigs obtained from short-reads assembled using metaSPAdes [22] and SGA [30] which represent the two assembly paradigms; de Bruijn graphs and overlap-layout-consensus (string graphs). The results showed that GraphBin2 achieves the best binning performance in both simulated and real datasets. Moreover, GraphBin2 shows the potential to infer contigs shared by multiple species. Note that GraphBin2 could be in principle applied to long-read assemblies. In the future, we intend to extend the capabilities of GraphBin2 to explore the avenues at improving the detection of contigs shared by multiple species and further extend towards binning long reads directly using read-overlap graphs.

References

1 Jarno Alanko, Fabio Cunial, Djamal Belazzougui, and Veli Mäkinen. A framework for space-efficient read clustering in metagenomic samples. *BMC Bioinformatics*, 18(3):59, March 2017. doi:10.1186/s12859-017-1466-6.

2 Johannes Alneberg, Brynjar Smári Bjarnason, Ino de Bruijn, Melanie Schirmer, Joshua Quick, Umer Z. Ijaz, Leo Lahti, Nicholas J. Loman, Anders F. Andersson, and Christopher Quince. Binning metagenomic contigs by coverage and composition. *Nature Methods*, 11:1144–1146, September 2014. doi:10.1038/nmeth.3103.

3 Sasha K. Ames, David A. Hysom, Shea N. Gardner, et al. Scalable metagenomic taxonomy classification using a reference genome database. *Bioinformatics*, 29(18):2253–2260, July 2013. arXiv:http://oup.prod.sis.lan/bioinformatics/article-pdf/29/18/2253/17128159/btt389.pdf.

4 Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, Alexey V. Pyshkin, Alexander V. Sirotkin, Nikolay Vyahhi, Glenn Tesler, Max A. Alekseyev,

and Pavel A. Pevzner. SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012. PMID: 22506599. doi:10.1089/cmb.2012.0021.

5 Tyler P. Barnum, Israel A. Figueroa, Charlotte I. Carlström, Lauren N. Lucas, Anna L. Engelbrektson, and John D. Coates. Genome-resolved metagenomics identifies genetic mobility, metabolic interactions, and unexpected diversity in perchlorate-reducing communities. *The ISME Journal*, 12(6):1568–1581, 2018. doi:10.1038/s41396-018-0081-5.

6 Joshua N. Burton, Ivan Liachko, Maitreya J. Dunham, and Jay Shendure. Species-level deconvolution of metagenome assemblies with hi-c–based contact probability maps. *G3: Genes, Genomes, Genetics*, 4(7):1339–1346, 2014. doi:10.1534/g3.114.011825.

7 Chon-Kit Kenneth Chan, Arthur L. Hsu, Saman K. Halgamuge, and Sen-Lin Tang. Binning sequences using very sparse labels within a metagenome. *BMC Bioinformatics*, 9(1):215, April 2008. doi:10.1186/1471-2105-9-215.

8 Brian Cleary, Ilana Lauren Brito, Katherine Huang, Dirk Gevers, Terrance Shea, Sarah Young, and Eric J. Alm. Detection of low-abundance bacterial strains in metagenomic datasets by eigengenome partitioning. *Nature Biotechnology*, 33:1053, September 2015. doi:10.1038/nbt.3329.

9 Garey, Michael R. and Johnson, David S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.

10 Samuele Girotto, Cinzia Pizzi, and Matteo Comin. MetaProb: accurate metagenomic reads binning based on probabilistic sequence signatures. *Bioinformatics*, 32(17):i567–i575, August 2016. doi:10.1093/bioinformatics/btw466.

11 Hadrien Gourlé, Oskar Karlsson-Lindsjö, Juliette Hayer, and Erik Bongcam-Rudloff. Simulating Illumina metagenomic data with InSilicoSeq. *Bioinformatics*, 35(3):521–522, July 2018. doi:10.1093/bioinformatics/bty630.

12 Damayanthi Herath, Sen-Lin Tang, Kshitij Tandon, David Ackland, and Saman Kumara Halgamuge. Comet: a workflow using contig coverage and composition for binning a metagenomic sample with high precision. *BMC Bioinformatics*, 18(16):571, December 2017. doi:10.1186/s12859-017-1967-3.

13 Dongwan Kang, Feng Li, Edward S Kirton, Ashleigh Thomas, Rob S Egan, Hong An, and Zhong Wang. MetaBAT 2: an adaptive binning algorithm for robust and efficient genome reconstruction from metagenome assemblies. *PeerJ*, 7:e27522v1, February 2019. doi:10.7287/peerj.preprints.27522v1.

14 David Kelley and Steven Salzberg. Clustering metagenomic sequences with interpolated Markov models. *BMC Bioinformatics*, 11(1):544, 2010. doi:10.1186/1471-2105-11-544.

15 Daehwan Kim, Li Song, Florian P. Breitwieser, and Steven L. Salzberg. Centrifuge: rapid and sensitive classification of metagenomic sequences. *Genome Research*, 26(12):1721–1729, 2016. arXiv:http://genome.cshlp.org/content/26/12/1721.full.pdf+html.

16 Cedric C. Laczny, Christina Kiefer, Valentina Galata, Tobias Fehlmann, Christina Backes, and Andreas Keller. BusyBee Web: metagenomic data analysis by bootstrapped supervised binning and annotation. *Nucleic Acids Research*, 45(W1):W171–W179, May 2017. doi:10.1093/nar/gkx348.

17 Heng Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem, 2013. arXiv:1303.3997.

18 Yunan Luo, Yun William Yu, Jianyang Zeng, Bonnie Berger, and Jian Peng. Metagenomic binning through low-density hashing. *Bioinformatics*, 35(2):219–226, July 2018. doi:10.1093/bioinformatics/bty611.

19 Vijini Mallawaarachchi, Anuradha Wickramarachchi, and Yu Lin. GraphBin: Refined binning of metagenomic contigs using assembly graphs. *Bioinformatics*, March 2020. btaa180. doi:10.1093/bioinformatics/btaa180.

20 Peter Menzel, Kim Lee Ng, and Anders Krogh. Fast and sensitive taxonomic classification for metagenomics with Kaiju. *Nature Communications*, 7:11257, April 2016. Article.

21 Eugene W. Myers. The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85, September 2005. doi:10.1093/bioinformatics/bti1114.

22 Sergey Nurk, Dmitry Meleshko, Anton Korobeynikov, and Pavel A. Pevzner. metaSPAdes: a new versatile metagenomic assembler. *Genome Research*, 27(5):824–834, 2017. doi:10.1101/gr.213959.116.

23 Rachid Ounit, Steve Wanamaker, Timothy J. Close, and Stefano Lonardi. CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics*, 16(1):236, March 2015. doi:10.1186/s12864-015-1419-2.

24 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001. doi:10.1073/pnas.171285098.

25 Christopher Quince, Alan W. Walker, Jared T. Simpson, Nicholas J. Loman, and Nicola Segata. Shotgun metagenomics, from sampling to analysis. *Nature Biotechnology*, 35(9):833–844, 2017. doi:10.1038/nbt.3935.

26 Christian S. Riesenfeld, Patrick D. Schloss, and Jo Handelsman. Metagenomics: Genomic analysis of microbial communities. *Annual Review of Genetics*, 38(1):525–552, 2004. PMID: 15568985. doi:10.1146/annurev.genet.38.072902.091216.

27 L Schaeffer, H Pimentel, N Bray, P Melsted, and L Pachter. Pseudoalignment for metagenomic read assignment. *Bioinformatics*, 33(14):2082–2088, February 2017. doi:10.1093/bioinformatics/btx106.

28 Karel Sedlar, Kristyna Kupkova, and Ivo Provaznik. Bioinformatics strategies for taxonomy independent binning and visualization of sequences in shotgun metagenomics. *Computational and Structural Biotechnology Journal*, 15:48–55, 2017. doi:10.1016/j.csbj.2016.11.005.

29 Itai Sharon, Michael J. Morowitz, Brian C. Thomas, Elizabeth K. Costello, David A. Relman, and Jillian F. Banfield. Time series community genomics analysis reveals rapid shifts in bacterial species, strains, and phage during infant gut colonization. *Genome Research*, 23(1):111–120, 2013. doi:10.1101/gr.142315.112.

30 Jared T. Simpson and Richard Durbin. Efficient de novo assembly of large genomes using compressed data structures. *Genome Research*, 22(3):549–556, 2012. doi:10.1101/gr.126953.111.

31 Marc Strous, Beate Kraft, Regina Bisdorf, and Halina Tegetmeyer. The Binning of Metagenomic Contigs for Microbial Physiology of Mixed Cultures. *Frontiers in Microbiology*, 3:410, 2012. doi:10.3389/fmicb.2012.00410.

32 Torsten Thomas, Jack Gilbert, and Folker Meyer. Metagenomics - a guide from sampling to data analysis. *Microbial Informatics and Experimentation*, 2(1):3, 2012. doi:10.1186/2042-5783-2-3.

33 Le Van Vinh, Tran Van Lang, Le Thanh Binh, and Tran Van Hoai. A two-phase binning algorithm using l-mer frequency on groups of non-overlapping reads. *Algorithms for Molecular Biology*, 10(1):2, January 2015. doi:10.1186/s13015-014-0030-4.

34 Jun Wang, Yuan Jiang, Guoxian Yu, Hao Zhang, and Haiwei Luo. BMC3C: binning metagenomic contigs using codon usage, sequence composition and read coverage. *Bioinformatics*, 34(24):4172–4179, June 2018. doi:10.1093/bioinformatics/bty519.

35 Ying Wang, Kun Wang, Yang Young Lu, and Fengzhu Sun. Improving contig binning of metagenomic data using d2S oligonucleotide frequency dissimilarity. *BMC Bioinformatics*, 18(1):425, September 2017. doi:10.1186/s12859-017-1835-1.

36 Ziye Wang, Zhengyang Wang, Yang Young Lu, Fengzhu Sun, and Shanfeng Zhu. SolidBin: improving metagenome binning with semi-supervised normalized cut. *Bioinformatics*, 35(21):4229–4238, April 2019. doi:10.1093/bioinformatics/btz253.

37 Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, 2014.

38 Yu-Wei Wu, Blake A. Simmons, and Steven W. Singer. MaxBin 2.0: an automated binning algorithm to recover genomes from multiple metagenomic datasets. *Bioinformatics*, 32(4):605–607, October 2015. doi:10.1093/bioinformatics/btv638.

39 Yu-Wei Wu, Yung-Hsu Tang, Susannah G. Tringe, Blake A. Simmons, and Steven W. Singer. Maxbin: an automated binning method to recover individual genomes from metagenomes using an expectation-maximization algorithm. *Microbiome*, 2(1):26, August 2014. doi:10.1186/2049-2618-2-26.

A Step-By-Step Example of Label Propagation in GraphBin2

Figure 3 shows how GraphBin2 propagates labels from Step 2 to Step 3 on the example assembly graph denoted in Figure 1. Figure 3(a) denotes the assembly graph after correcting labels of inconsistent vertices (after Step 2). In our example assembly graph that we have considered in Figure 1(a), the following candidate propagation actions will be executed in the given order.

(1) The candidate propagation action $(1, 0, 6, 1)$ is executed. Vertex 1 receives the red label from vertex 6 as shown in Figure 3(b).
(2) The candidate propagation action $(1, 0, 13, 14)$ is executed. Vertex 14 receives the red label from vertex 13 as shown in Figure 3(c).
(3) The candidate propagation action $(1, 1, 22, 21)$ is executed. Vertex 21 receives the green label from vertex 22 as shown in Figure 3(d).
(4) The candidate propagation action $(1, 2, 14, 7)$ is executed. Vertex 7 receives the red label from vertex 14 as shown in Figure 3(e).
(5) The candidate propagation action $(1, 3, 18, 19)$ is executed. Vertex 19 receives the blue label from vertex 18 as shown in Figure 3(f).
(6) The candidate propagation action $(1, 16, 8, 3)$ is executed. Vertex 3 receives the green label from vertex 8 as shown in Figure 3(g).
(7) The candidate propagation action $(1, 53, 21, 25)$ is executed. Vertex 25 receives the green label from vertex 21 as shown in Figure 3(h).

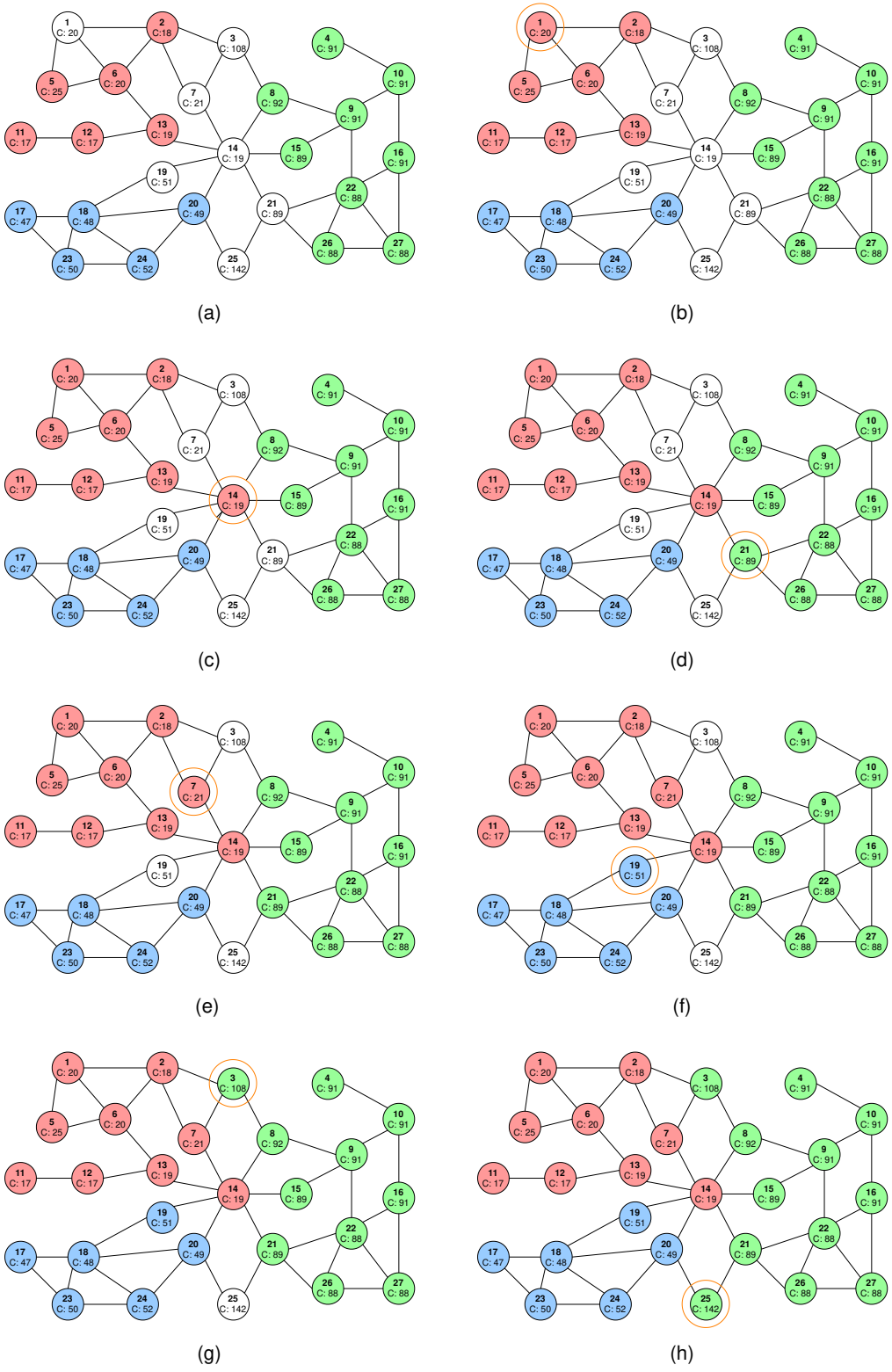

Figure 3 Step-by-step illustration of how labels are propagated in Step 3 of the GraphBin2 Workflow on the example assembly graph.

B Details on the Datasets

Table 11 Information on the datasets used for the experiments.

Dataset	Assembler	Read length (bp)	Number of paired end reads	Total number of non-isolated contigs	Mean contig length (bp)	Number of species in ground truth
Sim-5G	metaSPAdes	300	2,000,000	516	51,723	5
	SGA	300	2,000,000	18,192	1,675	5
Sim-10G	metaSPAdes	300	6,999,998	900	47,279	10
	SGA	300	6,999,998	32,389	1,300	10
Sim-20G	metaSPAdes	300	15,000,001	1,404	48,021	20
	SGA	300	15,000,001	72,791	873	20
Sharon-1	metaSPAdes	100	14,869,863	371	17,144	12
	SGA	100	14,869,863	766	3,034	12
Sharon-All	metaSPAdes	100	135,493,567	2,730	7,689	12
	SGA	100	135,493,567	20,942	1,547	12

C Commands Used

C.1 Assembly Tools

metaSPAdes

```
spades --meta -1 Reads_1.fastq -2 Reads_2.fastq -o /path/output_folder -t 20
```

SGA

```
sga preprocess -o reads.fastq --pe-mode 1 Reads_1.fastq Reads_2.fastq
sga index -a ropebwt -t 16 --no-reverse reads.fastq
sga correct -k 41 --learn -t 16 -o reads.k41.fastq reads.fastq
sga index -a ropebwt -t 16 reads.k41.fastq
sga filter -x 2 -t 16 reads.k41.fastq
sga fm-merge -m 45 -t 16 reads.k41.filter.pass.fa
sga index -t 16 reads.k41.filter.pass.merged.fa
sga overlap -m 55 -t 16 reads.k41.filter.pass.merged.fa
sga assemble -m 95 reads.k41.filter.pass.merged.asqg.gz
```

C.2 Binning Tools

MaxBin2

```
perl MaxBin-2.2.5/run_MaxBin.pl -contig contigs.fasta -abund abundance.abund -thread
8 -out /path/output_folder
```

Note: abundance.abund is a tab separated file with contig ID and the coverage for each contig in the assembly. metaSPAdes provides the coverage of each contig in the contig identifier of the final assembly. We can directly extract these values to create the abundance.abund file. However, no such information is provided for contigs produced by SGA. Hence, reads should be mapped back to contigs in order to determine the coverage of SGA contigs.

SolidBin

```
python scripts/gen_kmer.py /path/to/data/contig.fasta 1000 4
sh gen_cov.sh
```

```
python SolidBin.py --contig_file /path/to/contigs.fasta --composition_profiles
/path/to/kmer_4.csv --coverage_profiles /path/to/cov_inputtableR.tsv --output
/output/result.tsv --log /output/log.txt --use_sfs
```

GraphBin

metaSPAdes version

```
python graphbin.py --assembler spades --graph /path/to/graph_file.gfa --paths
/path/to/paths_file.paths --binned /path/to/binning_result.csv --output
/path/to/output_folder
```

SGA version

```
python graphbin.py --assembler sga --graph /path/to/graph_file.asqg --binned
/path/to/binning_result.csv --output /path/to/output_folder
```

GraphBin2

metaSPAdes version

```
python graphbin2.py --assembler spades --graph /path/to/graph_file.gfa --contigs
/path/to/contigs.fasta --paths /path/to/paths_file.paths --binned
/path/to/binning_result.csv --output /path/to/output_folder
```

SGA version

```
python graphbin2.py --assembler sga --graph /path/to/graph_file.asqg --contigs
/path/to/contigs.fasta --abundance /path/to/abundance.abund --binned
/path/to/binning_result.csv --output /path/to/output_folder
```