Graph-based self-supervised learning for repeat detection in metagenomic assembly

Ali Azizpour[1], Advait Balaji[2], Todd J. Treangen[2], and Santiago Segarra[1]

[1] Department of Electrical and Computer Engineering, Rice University, Houston, TX, USA
{aa210,segarra}@rice.edu
[2] Department of Computer Science, Rice University, Houston, TX, USA
{advait,treangen}@rice.edu

Running title: Graph-based self-supervised repeat detection

Abstract. Repetitive DNA (repeats) poses significant challenges for accurate and efficient genome assembly and sequence alignment. This is particularly true for metagenomic data, where genome dynamics such as horizontal gene transfer, gene duplication, and gene loss/gain complicate accurate genome assembly from metagenomic communities. Detecting repeats is a crucial first step in overcoming these challenges. To address this issue, we propose GraSSRep, a novel approach that leverages the assembly graph's structure through graph neural networks (GNNs) within a self-supervised learning framework to classify DNA sequences into repetitive and non-repetitive categories. Specifically, we frame this problem as a node classification task within a metagenomic assembly graph. In a self-supervised fashion, we rely on a high-precision (but low-recall) heuristic to generate pseudo-labels for a small proportion of the nodes. We then use those pseudo-labels to train a GNN embedding and a random forest classifier to propagate the labels to the remaining nodes. In this way, GraSSRep combines sequencing features with pre-defined and learned graph features to achieve state-of-the-art performance in repeat detection. We evaluate our method using simulated and synthetic metagenomic datasets. The results on the simulated data highlight our GraSSRep's robustness to repeat attributes, demonstrating its effectiveness in handling the complexity of repeated sequences. Additionally, our experiments with synthetic metagenomic datasets reveal that incorporating the graph structure and the GNN enhances our detection performance. Finally, in comparative analyses, GraSSRep outperforms existing repeat detection tools with respect to precision and recall.

Keywords: Metagenomics · Repeat detection · Graph neural network · Self-supervised learning · RECOMB24

Introduction

Metagenomics is a scientific discipline that involves analyzing genetic material obtained from complex uncultured samples housing DNA from diverse organisms (Wooley et al. 2010). This field utilizes high-throughput sequencing and bioinformatic techniques to characterize and compare the genomic diversity and functional potential of entire microbial communities without the need for isolating and culturing individual organisms (Yang et al. 2021). The resulting data can provide insights into the ecological roles and evolutionary relationships of the microorganisms present in the sample (Schatz et al. 2010).

However, the sequencing of DNA from such samples poses unique challenges. One of the major challenges in the metagenomic assembly is the presence of repeats (Ghurye et al. 2016, Lapidus and Korobeynikov 2021), which are sequences of DNA that are similar or identical to sequences elsewhere in the genome (Treangen and Salzberg 2012). The challenges posed by repeats in isolated genomes have primarily been addressed through the use of long-read technologies (Koren and Phillippy 2015). However, metagenomics presents a more complex problem as microbial mixtures often contain multiple closely related genomes that differ in just a few locations due to structural variants (Martin et al. 2023), such as horizontal gene transfer (Soucy et al. 2015), gene duplication, and gene loss/gain (Iranzo et al. 2019). Reads spanning the length of individual strains are required to fully resolve these genome-scale repeats present in microbiomes.

These repetitive elements, while natural and abundant in genomes, complicate the process of genome assembly and comparison (Treangen et al. 2009). They intricately tangle the assembly graph, making it difficult to distinguish the order, orientation, and copy number variation of genomes comprising the microbiome under study, resulting in fragmented assemblies. Moreover, repeats introduce ambiguities for comparative genomics, hindering differentiation between identical or similar regions and complicating the understanding of gene functions, regulatory elements, and their role in genetic disorders (Treangen and Salzberg 2012). To overcome these obstacles, precise identification and annotation of repeated sequences is necessary. Unraveling the complexities of repeated sequences is not only crucial for enhancing genome assembly but also essential for deciphering intricate regulatory mechanisms and evolutionary processes. Indeed, identifying these repeats is foundational for understanding genome stability, gene expression, and disease susceptibility, making the development of accurate repeat detection methods vital for advancing genomic research (Girgis 2015).

Graphs are powerful tools for visualizing complex relationships between various objects, such as DNA sequences. Graph-based algorithms can effectively represent the interconnections and overlapping patterns within genomes (Koutrouli et al. 2020), where the nodes in the graph represent unique DNA sequences. Due to the tangled nature of repeated sequences within the assembly graph, exploiting graph structure becomes particularly advantageous. As an illustrative example, Figure 1 portrays the assembly graph obtained from a simulated metagenome with two organisms. In this scenario, three random sequences are generated. Two of these sequences are inserted as intra-genome repeats in each organism, while the third one is inserted in both organisms, serving as an inter-genome repeat. This graph is visualized using Bandage (Wick et al. 2015), where the length of each node is proportional to the length of the corresponding contig. A node labeled as a repeat (which is colored red in the figure) represents a unique DNA sequence that occurs in several positions of the metagenome sample. The graph reveals that repeats are represented by central and well-connected nodes, indicating the potential of utilizing the inherent graph structure in genomic data for identifying repeated sequences. However, graph structure is usually not enough to tell apart the repeat nodes from some of the non-repeat ones. This motivates an approach that combines graph features with sequencing information such as read coverage or length of the DNA sequence.

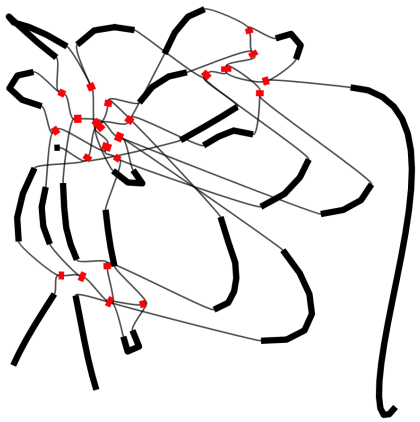

Figure 1. Assembly graph representation with repeat contigs in red.

Previous studies have employed pre-specified graph features in combination with machine learning techniques to address the challenge of detecting repeats, treating it as a node classification problem (Ghurye and Pop 2016, Ghurye et al. 2019). In this context, the nodes of the graph represent DNA sequences, and the

objective is to classify them into repeats and non-repeats. However, given the vast amount of genomic data, there remains ample opportunity for enhancement through learning discriminative graph features. One of the promising ways to achieve this is by employing graph neural networks (GNNs) (Wu et al. 2020). GNNs have the unique ability to learn distinctive and valuable features for the nodes within the graphs. Unlike predefined features, GNNs generate these characteristics through trainable iterative computations, making them adaptive to the specific data. These features have shown promising results in many other fields (Glaze et al. 2023, Čutura et al. 2021, Zhao et al. 2023, Chowdhury et al. 2021), emphasizing the efficiency of utilizing GNNs to classify nodes accurately and uncover the complexities within large graphs (Hamilton et al. 2017b).

However, one of the primary challenges in genomic data analysis is the fact that most of the data is unlabeled, particularly in distinguishing between repeat and non-repeat sequences. This characteristic of the data prevents the application of supervised or semi-supervised learning techniques for classifying DNA sequences (Kipf and Welling 2016). In the absence of labeled data points offering insights into each class, these conventional methods become ineffective. To overcome this issue, self-supervised learning emerges as a natural and powerful alternative to leverage the vast unsupervised data (Jaiswal et al. 2020). In self-supervised learning, specific data points (nodes) are initially given (potentially noisy) labels. Subsequently, machine learning algorithms are employed, coupled with fine-tuning steps, to refine the model's performance. This approach ensures the ability to classify data points without requiring access to their true labels.

In this paper, we propose GraSSRep, a novel graph-based algorithm to identify and detect the repeated sequences in the metagenomic assembly in a self-supervised manner. Our contributions are threefold: 1) By leveraging GNNs, we devise the first method that learns (rather than pre-specifies) graph features for repeat detection; 2) We establish the first algorithm that uses self-supervised learning for repeat detection, leveraging existing methods to generate noisy labels that we then refine and expand using our learnable architecture; 3) Through numerical experiments, we demonstrate the robustness of our methodology, the value of each of its steps, and the performance gain compared with the state of the art.

Methods

Given paired-end reads, our goal is to identify repeated DNA sequences in the metagenome (see Experimental setup for the precise criteria used to define a repeat). An overview of our method specifically designed for this task is illustrated in Figure 2. In the subsequent sections, we provide a detailed explanation of each step involved in the pipeline.

Step 1: Assembly graph construction

In the initial step, we construct an assembly graph in order to leverage graph features for repeat detection. To do so, we assemble the input reads to contigs and obtain the assembly graph as illustrated in Figure 2(A). Here, we use the popular metagenomic assembler, metaSpades (Nurk et al. 2017), which employs the multi-sized de Bruijn graph to derive the contigs and the connections between them. We consider these assembled contigs as the nodes $\mathcal{V}$ of our assembly graph, where $|\mathcal{V}| = N$. We denote by $\mathbf{A} \in \{0,1\}^{N\times N}$ the adjacency matrix of the corresponding unweighted graph, where $A_{ij} = 1$ if there is an edge between contig i and j, and $A_{ij} = 0$ otherwise. Note that assembly graphs can also be generated similarly using alternative metagenomic assemblers like MEGAHIT (Li et al. 2015) and metaFlye (Kolmogorov et al. 2020), either by assembling the contigs and connecting them using read-mapping information or by directly utilizing the assembly graph provided by the assembler. We plan to support these additional assembly graph formats in the future. We specifically choose metaSpades because it is a well-known state-of-the-art short read assembler that is easy to use and offers high accuracy. The most significant benefit of metaSpades is that it provides the assembly graph directly alongside the assembled contigs. This feature fits seamlessly into our pipeline and allows us to bypass the additional step of read-mapping to identify connections between contigs. This integrated process not only simplifies our workflow but also accelerates graph construction, making metaSpades the optimal choice for our framework. However, we can alternatively construct our assembly graph manually by assembling the reads and utilizing read-mapping data; this is discussed in further detail in Alternative graph construction in the Supplemental material.

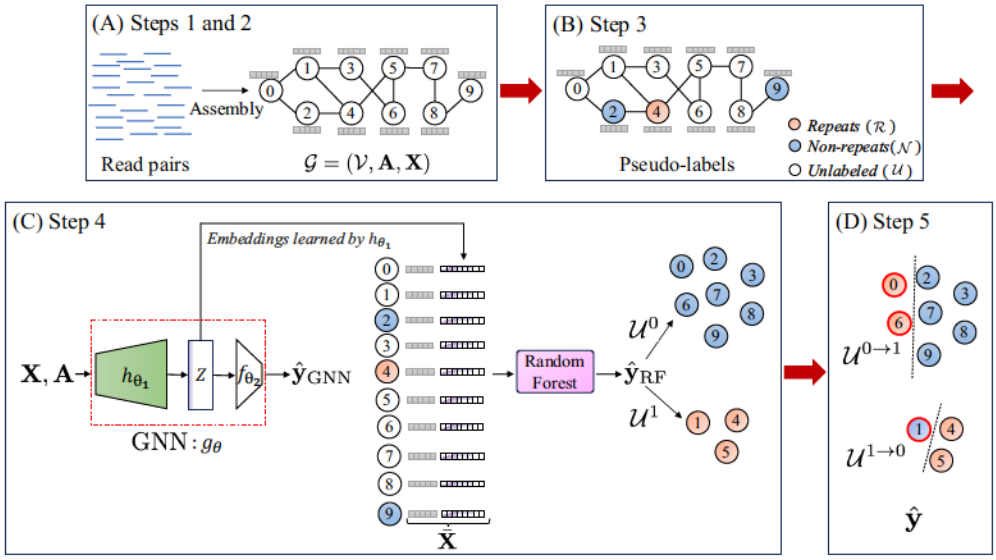

Figure 2. Overview of GraSSRep. (A) Reads are assembled into contigs, forming the nodes of the assembly graph. The graph structure (edges) is provided by metaSpades. Also, feature vectors are computed for each contig. (B) contigs with distinctive sequencing features are selected as training nodes and labeled. (C) The assembly graph is input into a GNN. Embeddings are generated for each contig and combined with the initial features. A random forest classifier predicts labels for all contigs based on the augmented feature vectors. (D) Sequencing features are employed to identify outliers within each predicted class, leading to the reassignment of their class labels.

Step 2: Feature extraction

We compute features of the contigs that are informative in determining which contigs are repetitive. We consider two types of features: sequencing and graph-based. Sequencing features (contig length and mean coverage) are obtained during the sequencing process before constructing the contig graph and used in Steps 3 and 5. In addition, we incorporate four graph-based features that are widely used in the literature: betweenness centrality, k-core value, degree, and clustering coefficient. Previous studies have emphasized the significance of betweenness centrality (Segarra and Ribeiro 2015) in identifying repeats (Ghurye and Pop 2016). Additionally, KOMB (Balaji et al. 2022) has underscored the crucial role of the k-core value in anomaly detection within contigs. Furthermore, the degree of nodes indicates their connectivity strength with other contigs, aiding in the identification of repeated regions. We also consider the clustering coefficient due to its substantial impact on node classification tasks, as well as its demonstrated positive effects and favorable outcomes in various related domains (Zaki et al. 2013). We store the graph-based features in a matrix $\mathbf{X} \in \mathbb{R}^{N\times 4}$, where every row contains the four graph-based features of a given node (contig) in the graph. Thus, we define our featured graph of interest $\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X})$ as shown in Figure 2(A).

Step 3: Selection of the training nodes

Recall that we do not have any prior information (labels) on whether any contig is a repeat or not. In this context, the idea of self-supervised learning is first to do a high-confidence classification of a subset of the contigs (assigning potentially noisy labels, denominated pseudo-labels, to a subset of the nodes) and then use those nodes as a training set for a machine learning model that can classify the remaining contigs. We generate this set of pseudo-labels using the sequencing features from Step 2. In generating pseudo-labels, it is important only to consider those for which we have a high level of confidence, so that the training process based on these pseudo-labels is reliable.

In defining our pseudo-labels, we rely on the fact that shorter contigs with higher coverage are highly likely to be repetitive, while very long contigs with lower coverage are more likely to be non-repeat contigs (Ghurye

and Pop 2016). More precisely, let us define as x_i^{len} and x_i^{cov} the length (number of base pairs) and coverage (mean number of reads mapped to the base pairs in the contig) of node i, respectively. We set a percentile p (with $0 \leq p \leq 50$) based on which we define the following thresholds: $\tau_{\text{low}}^{\text{len}}$ is the p-th percentile of the lengths among all contigs in $\mathcal{V}$, $\tau_{\text{high}}^{\text{len}}$ is the $(100-p)$-th percentile of the lengths among all contigs, and τ^{cov} is the $(100-p)$-th percentile of the coverages among all contigs. Based on these thresholds, we divide the contigs into three sets, the repeats $\mathcal{R}$, the non-repeats $\mathcal{N}$, and the unlabeled $\mathcal{U}$, as follows

$$\mathcal{R} = \{i \in \mathcal{V} \,|\, x_i^{\text{len}} < \tau_{\text{low}}^{\text{len}} \wedge x_i^{\text{cov}} > \tau^{\text{cov}}\}, \qquad \mathcal{N} = \{i \in \mathcal{V} \,|\, x_i^{\text{len}} > \tau_{\text{high}}^{\text{len}} \wedge x_i^{\text{cov}} < \tau^{\text{cov}}\}, \tag{1}$$

and $\mathcal{U} = \mathcal{V} \setminus (\mathcal{R} \cup \mathcal{N})$. In (1), contigs shorter than the lower length threshold and with a coverage surpassing the coverage threshold are included in the training set with a repeat pseudo-label ($\mathcal{R}$). Conversely, contigs exceeding the higher length threshold and having a coverage below the coverage threshold are added to the training set with a non-repeat pseudo-label ($\mathcal{N}$). If a contig does not meet any of these conditions, it suggests that sequencing features alone are not sufficient to determine its classification. Consequently, these contigs are not included in the training set ($\mathcal{U}$). A simple example of how the assembly graph is divided into three subsets after this step is depicted in Figure 2(B).

Step 4: contig classification via self-supervised learning

We leverage self-supervised learning by training a graph-based model on $\mathcal{R}$ (binary label of 1) and $\mathcal{N}$ (binary label of 0) and use that model to classify the nodes in $\mathcal{U}$.

Consider the graph $\mathcal{G} = (\mathcal{V}, \mathbf{A}, \mathbf{X})$ generated in Steps 1 and 2 and denote by g_θ a GNN parameterized by θ (Wu et al. 2020). This GNN takes the graph structure $\mathbf{A}$ and the node features $\mathbf{X}$ as input and produces labels $\hat{\mathbf{y}}_{\text{GNN}}$ for the nodes at the output. To generate these labels, g_θ can be viewed as an end-to-end network that is structured as follows

$$\hat{\mathbf{y}}_{\text{GNN}} = g_\theta(\mathbf{X}, \mathbf{A}) = f_{\theta_2}(h_{\theta_1}(\mathbf{X}, \mathbf{A})), \tag{2}$$

where h_{θ_1} consists of graph convolutional layers followed by an activation function (Agarap 2018). Each layer in h_{θ_1} generates new observations for every node based on its neighboring nodes. These convolutional layers are succeeded by f_{θ_2}, which represents a fully connected neural network (Haykin 1998). The purpose of this network is to predict the final label for each node based on the features derived from the last layer of h_{θ_1}. Note that we provide here a generic functional description of our methodology whereas in Experimental setup, we detail the specific architecture used in the experiments.

We denote the output of the convolutional layers by $\mathbf{Z} = h_{\theta_1}(\mathbf{X}, \mathbf{A}) \in \mathbb{R}^{N \times d}$, where d is a pre-specified embedding dimension. The i-th row $\mathbf{z}_i$ of $\mathbf{Z}$ represents new features for contig i, learned in such a way that the final linear layer, f_{θ_2}, can predict the class of the contigs based on these features. These embeddings enable us to achieve our objective of understanding the graph-based characteristics of repeat and non-repeat contigs. Notice that the features in $\mathbf{z}_i$ not only depend on graph features of node i but also on the features of its local neighborhood through the aggregation of the trainable convolutional layers in h_{θ_1}.

In order to learn the parameters $\theta = \{\theta_1 \cup \theta_2\}$, the GNN undergoes an end-to-end training based on the pseudo-labels $\mathcal{R}$ and $\mathcal{N}$ identified in Step 3. This training process involves minimizing a loss function that compares the predicted labels $\hat{\mathbf{y}}_{\text{GNN}}$ with the pseudo-labels

$$\theta^\star = \underset{\theta}{\text{argmin}} \sum_{i \in \mathcal{R}} \mathcal{L}([\hat{\mathbf{y}}_{\text{GNN}}(\theta)]_i, 1) + \sum_{i \in \mathcal{N}} \mathcal{L}([\hat{\mathbf{y}}_{\text{GNN}}(\theta)]_i, 0), \tag{3}$$

where $\mathcal{L}$ represents a classification loss (such as cross-entropy loss (De Boer et al. 2005)) and we have made explicit the dependence of $\hat{\mathbf{y}}_{\text{GNN}}$ with θ. In essence, in (3) we look for the GNN parameters $\theta^\star$ such that the predicted labels for the nodes in $\mathcal{R}$ are closest to 1 while the predicted labels for the nodes in $\mathcal{N}$ are closest to 0. Intuitively, the intermediate embeddings $\mathbf{Z}$ obtained using the optimal parameters $\theta^\star$ encode learning-based features relevant for the classification beyond the pre-defined ones in Step 2. Thus, we construct the augmented feature matrix $\bar{\mathbf{X}} = [\mathbf{X}, \mathbf{Z}] \in \mathbb{R}^{N \times (4+d)}$ by concatenating the initial graph-based features with those generated by the GNN.

A random forest (RF) classifier is then trained on the pseudo-labels $\mathcal{R} \cup \mathcal{N}$ having the augmented features $\bar{\mathbf{X}}$ as input. The RF is trained by creating multiple decision trees from different subsets of the dataset (a process known as bootstrapping), with each tree using a random subset of features. When making predictions,

the individual trees' outputs are combined through majority voting, producing a reliable and precise ensemble model (Breiman 2001). The RF classifier combines the explanatory power of the original graph-based features $\mathbf{X}$ found to be relevant in previous works with the learning-based features $\mathbf{Z}$ to generate the predicted labels $\hat{\mathbf{y}}_{\text{RF}}$. An overview of how the labels of the training contigs are propagated to all contigs in Step 4 is shown in Figure 2(C).

Notice that the sequencing features x^{len} and x^{cov} are not used in computing $\hat{\mathbf{y}}_{\text{RF}}$ other than in the generation of the pseudo-labels. If we were to include these features as inputs to the RF, then the classifier can simply learn the conditions in (1) and obtain zero training error by ignoring all the graph features. This would directly defeat the purpose of our self-supervised framework. Instead, the current pipeline can distill the graph-based attributes associated with repeats and non-repeats, enabling us to generalize this knowledge to classify other contigs effectively.

Step 5: Fine-tuning the labels

In the final step of our method, we enhance the performance of our predictions through a fine-tuning process. We first assign the pseudo-labels of the training nodes in $\mathcal{R}$ and $\mathcal{N}$ as their final predicted labels. Our primary focus is then directed toward the non-training contigs in $\mathcal{U}$. These contigs have been classified by the RF in Step 4 relying solely on their graph-based features and embeddings learned by the GNN. At this point, reconsidering sequencing features becomes crucial, as they hold valuable information that can significantly contribute to determining the accurate labels of the contigs.

To do so, we divide the contigs in $\mathcal{U}$ into two disjoint sets: those predicted as repeats (label 1) by $\hat{\mathbf{y}}_{\text{RF}}$ form the set $\mathcal{U}^1$ and those predicted as non-repeats (label 0) by $\hat{\mathbf{y}}_{\text{RF}}$ form the set $\mathcal{U}^0$. Within each set, our objective is to identify outliers using the sequencing features x^{len} and x^{cov} and modify their labels accordingly, similar to Step 3. Within each set, specific thresholds are computed based on the distribution of sequencing features of the contigs in that set. More precisely, for $\mathcal{U}^1$ we define $\rho^{\text{len}}_{\text{high}}$ as the $(100-p)$-th percentile of the contigs' lengths and $\rho^{\text{cov}}_{\text{low}}$ and the p-th percentile of the coverage. Conversely, for $\mathcal{U}^0$ we define $\rho^{\text{len}}_{\text{low}}$ as the p-th percentile of the contigs' lengths and $\rho^{\text{cov}}_{\text{high}}$ and the $(100-p)$-th percentile of the coverage. Based on these thresholds, we identify outliers based on the following criteria

$$\mathcal{U}^{1\to 0} = \{i \in \mathcal{U}^1 \,|\, x_i^{\text{len}} > \rho^{\text{len}}_{\text{high}} \ \wedge \ x_i^{\text{cov}} < \rho^{\text{cov}}_{\text{low}}\}, \qquad \mathcal{U}^{0\to 1} = \{i \in \mathcal{U}^0 \,|\, x_i^{\text{len}} < \rho^{\text{len}}_{\text{low}} \ \wedge \ x_i^{\text{cov}} > \rho^{\text{cov}}_{\text{high}}\}. \tag{4}$$

In (4), we change the label from repeat to non-repeat ($\mathcal{U}^{1\to 0}$) for those contigs that are longer than a threshold and have low coverage. Similarly, we change the label from non-repeat to repeat ($\mathcal{U}^{0\to 1}$) for short contigs with high coverage. This process is illustrated in Figure 2(D). Notice that we used the same percentile p to compute the thresholds ρ here as that one used to compute the thresholds τ in Step 3. Naturally, we could select a different percentile here, but we use the same one as this shows good empirical results and reduces the number of hyperparameters.

Summarizing, the final labels $\hat{\mathbf{y}}$ predicted by our model are given by

$$[\hat{\mathbf{y}}]_i = \begin{cases} 1 & \text{for all } \ i \in \mathcal{R} \cup (\mathcal{U}^1 \setminus \mathcal{U}^{1\to 0}) \cup \mathcal{U}^{0\to 1}, \\ 0 & \text{for all } \ i \in \mathcal{N} \cup (\mathcal{U}^0 \setminus \mathcal{U}^{0\to 1}) \cup \mathcal{U}^{1\to 0}. \end{cases} \tag{5}$$

In (5), we see that the contigs deemed as repeats ($[\hat{\mathbf{y}}]_i = 1$) by our method are those i) assigned a repeat pseudo-label in Step 3 ($\mathcal{R}$), ii) classified as repeats by our RF in Step 4 and not deemed as outliers in Step 5 ($\mathcal{U}^1 \setminus \mathcal{U}^{1\to 0}$), or iii) classified as non-repeats in Step 4 but later deemed as outliers in Step 5 ($\mathcal{U}^{0\to 1}$). Conversely, contigs classified as non-repeats are those i) assigned a non-repeat pseudo-label in Step 3 ($\mathcal{N}$), ii) classified as non-repeats by our RF in Step 4 and not deemed as outliers in Step 5 ($\mathcal{U}^0 \setminus \mathcal{U}^{0\to 1}$), or iii) classified as repeats in Step 4 but later deemed as outliers in Step 5 ($\mathcal{U}^{1\to 0}$).

Results

In the following sections, we present a comprehensive analysis of our algorithm's performance across various settings.

Experimental setup

Datasets We test GraSSRep in three types of datasets.

Simulated data: To represent distinct organisms, we generate two random backbone genomes with an equal probability of observing each base. Subsequently, a random sequence of length L is generated for each backbone and integrated into the genome with a copy number of C, serving as an intra-genome repeat. Additionally, an inter-genome repeat of length L is randomly generated and inserted C times in both genomes, representing an inter-genome repeat. Unlike the backbone genomes, repeats exhibit a non-uniform distribution of bases, resulting in distinctive characteristics unique to each repeat, setting them apart from the backbone genome. Consequently, we have two genomes, both containing a repeat content of $2 \times L \times C$ within a fixed length of 5 million base pairs for each organism. As a result, the characteristics of the repeats within the genomes can be controlled by adjusting the values of L and C. Finally, simulated reads, each 101 base pairs in length, are generated using wgsim (https://github.com/lh3/wgsim) with default values for error (2%) and mutation (0.1%).

Shakya 1: In this dataset, we analyze the reference genomes of a synthetic metagenome called Shakya, which consists of 64 organisms, including 48 bacteria and 16 archaea (Shakya et al. 2013). Based on these reference genomes, read pairs are generated using wgsim, akin to the previous dataset. However, unlike the simulated data, all the backbone genomes in this dataset are real organisms, containing intricate repeat patterns that are beyond our control. The generated reads are 101 base pairs long with a high coverage ($\simeq 50$), and are produced without any errors or mutations, in order to identify exact repeats in the data.

Shakya 2: Read pairs from the Shakya (Shakya et al. 2013) study were obtained from the European Nucleotide Archive (ENA – Run:SRR606249), all with a length of 101. We have no influence over coverage or read errors in this set of reads, mirroring real-world settings. This characteristic enables us to evaluate GraSSRep under realistic scenarios.

Assembly In all experiments, contigs are assembled using the default values of metaSpades v3.13.0 for k-mer size, which are $k = 21$, $k = 33$, and $k = 55$. Also, in the error-free case (Shakya 1 dataset), we utilize the `--only-assembler` option of metaSpades and disable the read error correction step.

To assess our model accurately, it is crucial to have the ground truth labels for the contigs. To identify these labels, all contigs are aligned to the reference genomes using NUCmer (Marçais et al. 2018) (with the `--maxmatch` option). Contigs are marked as repeats if they meet specific criteria. Generally, this criterion includes aligning at more than one location with at least 95% identity and 95% alignment length, indicating non-identical repeats. However, in error-free cases like the Shakya 1 dataset, the criterion is aligning at more than one location with 100% identity and 100% alignment length, which indicates exact repeats through the reference genomes.

Method design and hyperparameter choices To select and label the training nodes, a threshold value p ranging between 30 and 40 is employed in Step 3, depending on the presence of noise in the data. Specifically, $p = 35$ in instances where noise is present (simulated data and Shakya 2), ensuring robustness in the presence of data irregularities. However, for noiseless cases (Shakya 1), we set $p = 20$, leading to a stricter definition of repeat pseudo-labels. Previous studies have demonstrated that this choice yields effective repeat detection (Ghurye et al. 2019). However, in the simulated dataset, during the fine-tuning step, we observed that setting $p = 0$ (indicating no need for fine-tuning) yielded superior results. This phenomenon primarily arises due to the presence of only two organisms in the dataset, leading to smaller and simpler assembly graphs. Consequently, the fine-tuning step becomes unnecessary as the labels generated by RF suffice for accurate classification.

In Step 4, the first component of the GNN, h_{θ_1}, consists of two consecutive GraphSAGE convolutional layers, each followed by a ReLU activation function (Hamilton et al. 2017a). The node representation update in these layers can be mathematically defined as follows:

$$\mathbf{z}_v^{(l+1)} = \mathrm{ReLU}\left(\left[\mathbf{W}_k \cdot \mathrm{Mean}\left(\left\{\mathbf{z}_u^{(l)}, \forall u \in \mathrm{Neigh}(v)\right\}\right), \mathbf{B}_k \mathbf{z}_v^{(l)}\right]\right), \quad \forall v \in \mathcal{V},$$

where $\mathbf{z}_v^{(l)}$ represents the node embedding of the node v at layer l, $\mathrm{Neigh}(v)$ represents the set of neighboring nodes of node v, and Mean is an aggregation function that combines the embeddings of neighboring nodes.

Moreover, $\mathbf{B}_k$ and $\mathbf{W}_k$ represent the linear transformation matrix for the self and neighbor embeddings, respectively. In this equation, $\mathbf{z}_v^{(l+1)}$ represents the updated embedding of the node v at the next layer $(l+1)$. Both the first and second convolutional layers have 16 hidden channels. More details on tuning the GNN structure hyperparameters are provided in GNN hyperparameter tuning in the Supplemental material. This results in $d = 16$ new features being generated for each node, represented as $\mathbf{Z} \in \mathbb{R}^{N\times 16}$. Since h_{θ_1} has two graph convolutional layers, the final embeddings combine the features within the 2-hop neighborhoods of each node. Additionally, the second component of the GNN, f_{θ_2}, comprises a single fully connected layer that transforms the newly learned features, $\mathbf{Z}$, into binary classes using a linear transformation matrix $\mathbf{T} \in \mathbb{R}^{16\times 2}$. The GNN is trained for 2000 epochs, utilizing cross-entropy as the loss function and employing the Adam optimizer (Kingma and Ba 2014) with a learning rate of 0.001.

The RF classifier utilizes 100 trees in the forest to generate its results. The split criterion for each decision tree is determined using the Gini impurity measure, ensuring the creation of optimal splits at each node. Finally, to account for the randomness inherent in the training process, both the training and testing steps are repeated for 10 iterations in each case. The reported results are averaged across these iterations, providing a robust and reliable evaluation. As figures of merit, we report the classification accuracy, precision, recall, and F1-score (harmonic mean of precision and recall).

Evaluation on varying repeat characteristics

We leverage the simulated dataset introduced in Experimental setup to examine the effect of three crucial characteristics that are beyond our control within the real datasets:

A) Length of the repeats. To measure the impact of repeat length, we fix the copy number of both inserted intra-genome and inter-genome repeats at $C = 25$ and vary their length from $L = 150$ to $L = 1000$ base pairs, leading to a copy content ranging from 0.15% to 1% in the reference genomes.

B) Copy number of the repeats. We set the length of the inserted repeats to $L = 400$ base pairs and adjust their copy number from $C = 10$ to $C = 150$, increasing the complexity of the dataset. This results in a copy content ranging between 0.16% and 2.4% in the reference genomes.

C) Coverage. We generate backbone data by inserting repeats of $L = 400$ base pairs in length with a copy number of $C = 25$ to have 0.4% copy content in the reference genomes. The number of generated read pairs is varied, ranging from 0.25 to 2.5 million base pairs. Consequently, the coverage ranges from 5 to 50, allowing us to analyze the algorithm's performance under different coverage levels.

These adjustments enable a detailed evaluation of our algorithm's robustness and adaptability across a spectrum of repeat characteristics and coverage scenarios. Note that due to errors and mutations in the generated reads, our analysis considers a repeat as having at least 95% identity over 95% of the length. Consequently, more than just three contigs are identified as repeats in this context, each with copy numbers that may differ from the exact number of inserted repeats.

Since the backbone and inserted repeats are generated randomly in the simulated datasets, we conduct 10 trials for each case to ensure robust results for each condition. Specifically, for each scenario, we generate 10 datasets with the same desired characteristics for repeat length, copy number, or coverage. We then calculate the results for each trial and report the average across these trials for all metrics. Additionally, the figures depict the error for the F1 Score across these 10 trials as a shaded purple area. We use the interquartile range to quantify the error, i.e., the error lower bound corresponds to the 25th percentile, and the upper bound corresponds to the 75th percentile across the 10 samples.

As illustrated in Figure 3(A), our approach demonstrates resilience to variations in repeat length, with all metrics remaining stable as the repeat length increases. Consistently achieving an average F1-score above 99% indicates that our approach can effectively detect repeated contigs even when longer repeats are present in the dataset.

Figure 3(B) shows the performance attained when varying the copy number. Our method consistently achieves an average F1-score exceeding 97%, and for copy numbers below 70, it consistently surpasses 99%. Additionally, the average precision is higher than 99% in almost all cases. However, we observe a decreasing trend in the average recall, which results in a corresponding decrease in the F1-score as the copy number increases. This drop occurs because a higher copy number for the repeats creates a more tangled assembly graph with a lot of connections between the assembled contigs, making it more challenging to detect all the

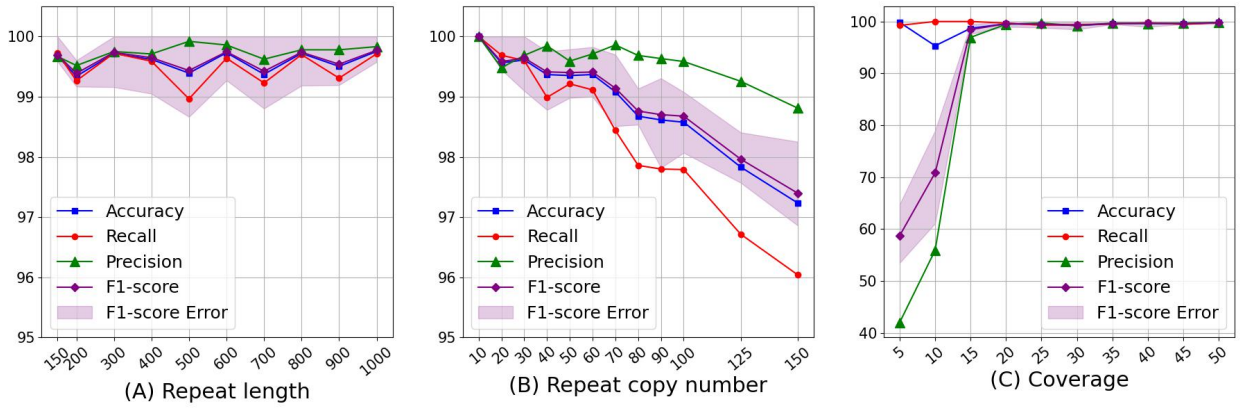

Figure 3. Assessing the method across various repeat characteristics. (A) The model remains stable in metrics even with increasing repeat length. (B) The method is robust to the copy number variation, consistently achieving an F1-score above 90%. (C) Higher sequencing coverage improves the model's performance.

repeats. Note that for copy numbers less than 10, the assembly graph remains untangled, and repeats are detected with 100% accuracy using coverage or degree without requiring additional complex steps.

As demonstrated in Figure 3(C), the model's performance exhibits a constant improvement with increased coverage, as expected. Specifically, when coverage is higher than 20 (corresponding to 1 million reads), the model achieves an almost perfect rate of nearly 100% for all metrics.

Ablation study of the steps of the algorithm

We focus on the behavior of our method (see 'Methods' Section) across different steps using the Shakya 1 dataset. After assembling and constructing the graph, we have $N = 51549$ contigs as the nodes of the graph, out of which 13842 contigs are exact repeats (total length of the contig repeated with 100% identity).

To begin, our evaluation involves assessing the method across various steps of the pipeline. Specifically, we examine the outcomes relative to the baseline, the results produced by the GNN ($\hat{\mathbf{y}}_{\text{GNN}}$), the outputs generated by RF ($\hat{\mathbf{y}}_{\text{RF}}$), and finally, after the fine-tuning step ($\hat{\mathbf{y}}$). In this context, the term "baseline" refers to a straightforward heuristic used to classify the contigs. This heuristic relies on Step 3 and labels nodes according to the following criteria

$$[\hat{\mathbf{y}}_{\text{base}}]_i = \begin{cases} 1 & \text{for all } \; i \in \mathcal{R}, \\ 0 & \text{for all } \; i \in \mathcal{N} \cup \mathcal{U}. \end{cases} \tag{6}$$

This approach allows us to test the effectiveness of sequencing features in node labeling in the absence of graph-based features.

In Figure 4(A), it is evident that the F1-score consistently rises throughout the pipeline, emphasizing the importance of each step in achieving optimal results. The baseline method exhibits high precision (98.3%) but low recall (40.9%), indicating appropriate node selection for determining pseudo-labels but an inability to identify most repeats. This observation underscores that sequencing features alone are insufficient for detecting repeats. This limitation is modified by the GNN, which significantly boosts the recall to 68.6%, effectively identifying more repeats, which suggests that graph structure is significant in detecting the repeats. Subsequent application of the RF further amplifies this increase in recall to 80.2%. However, this enhanced recall comes at the cost of reduced precision compared to the baseline. To address this precision loss, the fine-tuning step effectively identifies outliers, leading to a precision increase from 72.6% at the output of the RF to 83.8% for the final estimation. In summary, our approach yields a 88.9% F1-score without any prior labels on the contigs, representing a substantial improvement of 31.2% over the baseline method.

Moreover, we investigate the impact of the GNN and the embeddings it generates. To assess this, we perform two analyses. First, we exclude $\mathbf{Z}$ from the feature matrix fed to the RF, resulting in $\bar{\mathbf{X}} = [\mathbf{X}] \in \mathbb{R}^{N \times 4}$

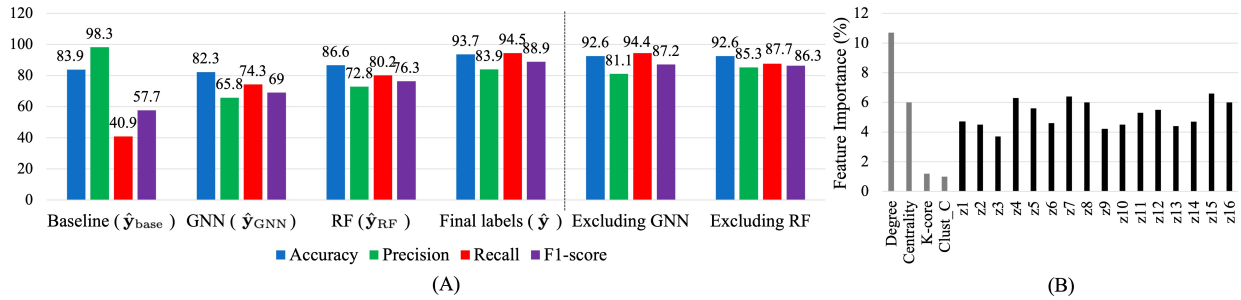

Figure 4. Behavior of GraSSRep across different steps. (A) Progression of the method's performance throughout the different steps, highlighting the effectiveness of each step in improving repeat detection. We also test the impact of excluding the GNN embeddings and RF step applied to the augmented feature vectors. (B) High importance of GNN-generated embeddings in RF classification.

aiming to observe the method's performance only based on the initial graph-based features. As depicted in Figure 4(A) under 'Excluding GNN', this exclusion leads to a decrease in all performance metrics. This decline suggests that embeddings play a crucial role in enhancing the reliability of repeat detection. Second, we calculate the importance of the features fed to the RF by averaging the impurity decrease from each feature across trees. The more a feature decreases the impurity, the more important it is. These importance values are then plotted in Figure 4(B). The plot indicates that all learned embeddings (labeled z1 through z16) exhibit high importance. This finding emphasizes the utility of the embeddings generated by the GNN in improving the overall performance of the method. Further discussion on the effect of the GNN can be found in GNN effect in the Supplemental material. Moreover, by removing the intermediate RF step and directly applying the fine-tuning process to the GNN-generated labels, we evaluate the effect of the RF step. As illustrated in Figure 4(A) under 'Excluding RF', this omission also results in a decrease in all performance metrics, highlighting the essential role of the RF in balancing the influence of initial features and the learned embeddings.

Additionally, we perform an ablation study on the percentile value p used to define the thresholds in Steps 3 and 5. The analysis in Ablation study on the percentile value p in the Supplemental material reveals that our approach is robust to this hyperparameter, particularly within the range of 30 to 40, which corresponds to the range used in our experiments.

Lastly, if we replicate the analysis in Figure 4 with an alternative graph construction method, we observe that all outcomes align consistently as outlined in Alternative graph construction in the Supplemental material. This illustrates the versatility of our tool, demonstrating its efficacy across diverse graph structures.

Comparison with existing repeat detection methods

We present a comprehensive comparison of our method with several existing repeat detection methods using contigs assembled from the reads downloaded from ENA (Shakya 2). The ground truth labels are obtained in the same manner as described in Experimental setup, using the reference genomes from the Shakya 1 dataset.

We consider five widely recognized methods for this comparison. Opera (Gao et al. 2011) and SOPRA (Dayarian et al. 2010) identify repetitive contigs by filtering out those with coverage 1.5 and 2.5 times higher than the average coverage of all contigs, respectively, without considering any graph structure. Similarly, the MIP scaffolder (Salmela et al. 2011) utilizes both high coverage (more than 2.5 times the average) and a high degree (≥ 50) within the assembly graph to detect the repeats. However, as the degree of contigs in the graph provided by metaSpades typically does not reach 50, we utilize an adaptive approach. In this alternative, we adjust the threshold from 50 to the 75-th percentile of the degrees observed in the graph. Additionally, Bambus2 (Koren et al. 2011) categorizes a contig as a repeat if the betweenness centrality, divided by the contig length, exceeds the upper bound of the range within c standard deviations above the mean on this feature. Here, c represents a hyperparameter of this method, and the optimal outcome on our

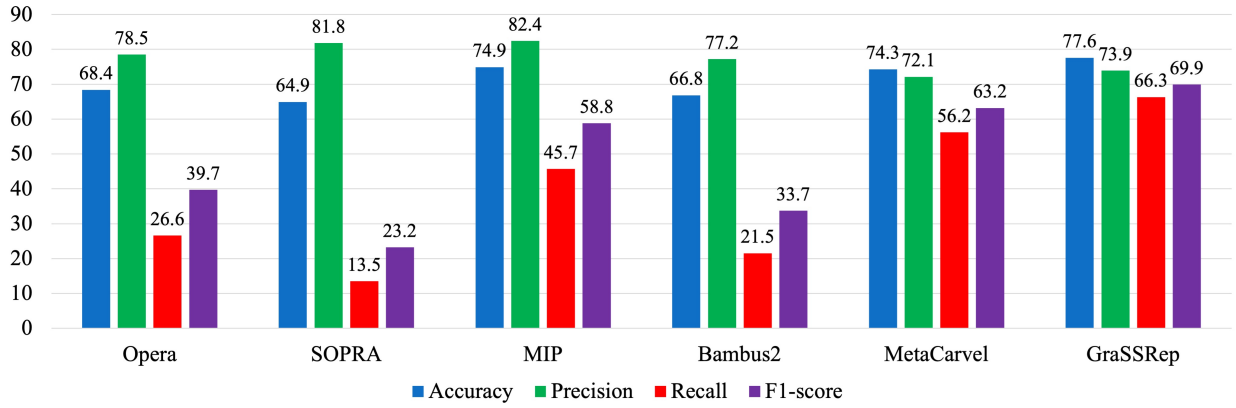

Figure 5. GraSSRep compared to the other repeat detection methods.

dataset was achieved with $c = 0$. Lastly, Metacarvel (Ghurye et al. 2019) employs four more complex graph-based features alongside coverage in a two-step process. First, any contig with a high betweenness centrality ($\geq$ three standard deviations plus the mean) on the assembly graph is marked as a repeat. Moreover, a contig is identified as a repeat if it falls within the upper quartile for at least three of these features: mean coverage, degree, ratio of skewed edges (based on coverage), and ratio of incident edges invalidated during the orientation phase of the contigs; see (Ghurye et al. 2019) for details. Notably, since we utilize a contig graph instead of a scaffold graph, we do not incorporate the latest feature and adjust the flag threshold from three to two in the second step.

As illustrated in Figure 5, GraSSRep outperforms all other methods, particularly demonstrating superior capability in detecting repeats with a higher recall rate (66.3% versus the next best alternative at 56.2%).

Thus far, we have focused on the practical unsupervised setting where no repeat labels are available. For completeness, we now consider the case where repeat labels for some contigs are available. This setting might arise, e.g., if we have knowledge about specific organisms present in the metagenomic sample and their corresponding reference genomes are accessible. GraSSRep can seamlessly accommodate this case. In our pipeline, we can leverage this prior knowledge to substitute Step 3. Instead of pseudo-labels, we employ the known node labels as our training set, leading to a semi-supervised (instead of self-supervised) setting. Our analysis in Incorporating prior knowledge in the Supplemental material shows that performance can be markedly improved in the case where labels are available for a fraction of the contigs.

Discussion

We tackled the challenging task of detecting repetitive sequences (repeats) in metagenomics data when we only have access to paired-end reads. We introduced GraSSRep, a novel method that leverages the inherent structure of the assembly graph by employing GNNs to extract specific features for the contigs. Moreover, adopting a self-supervised learning framework, we generated noisy pseudo-labels for a subset of the contigs, which were then used to train a graph-based classifier on the rest of the contigs.

Experimental studies using simulated datasets demonstrated the robustness of GraSSRep across diverse repeat characteristics and its resilience not only to repeat length but also to copy number variations. This ensures its applicability across various datasets and scenarios. Moreover, using synthetic datasets, we show the value of every step in our algorithm in enhancing repeat detection performance. This highlights the importance of each step and its role in achieving the best results. Furthermore, the GNN step effectively learns distinctive and important features for the repeat detection task based on the dataset, thereby enhancing the pipeline's ability to detect more repeats using the graph structure.

Additionally, we observed performance gain compared to existing repeat detection tools. This superiority comes from the combined value of incorporating learnable graph features (through the GNN) and considering a self-supervised framework. Notice that even if we fix the embedding dimension at $d = 16$, the graph features learned by the GNN depend on the specific dataset under consideration. In this way, our trainable architecture

can distill the key graph features that characterize repeats in the specific metagenomic sample. This adaptive approach stands in contrast to other methods, which often rely on fixed features. Moreover, since the RF is not pre-trained but rather trained based on the pseudo-labels, different features may vary in importance based on context. In this way, our self-supervised framework allows us to adapt to the metagenomic data at hand, and we do not have to worry about generalization issues of pre-trained models.

One limitation of our work is its dependence on the initial pseudo-labels. Specifically, in order to effectively generalize the labels from the initial training set to the other unlabeled nodes, we need sufficient samples in both repeat and non-repeat sets of training contigs from a diverse set of organisms. However, this process can be hindered by unbalanced coverage across different organisms. When some organisms exhibit significantly higher coverage compared to the rest of the community, the contigs generated from these organisms tend to dominate the high percentile of coverage and are detected as repeats in Step 3. Consequently, our training set becomes biased towards a few organisms, impeding the detection of repeats of other organisms. To address this issue, we plan to develop a more systematic approach to training set selection in future work.

Furthermore, it is worth noting that while we selected the indicated reference genomes for the Shakya community for ground truth detection and evaluation of our method, previous studies (Ondov et al. 2019) have identified additional reference genomes present in the community. Consequently, some repeats may be missing from the ground truth set, as their reference genomes are not included in the community and thus not identified as true repeats.

A natural extension of our approach is its integration into widely used assemblers. This integration would replace their existing repeat detection modules with GraSSRep, yielding potential improvements in assembly quality. We also intend to apply our method to real datasets, particularly in environments like hot springs where widely accessible reference genomes are scarce. Lastly, the overall pipeline of GraSSRep can potentially address other problems in genomics where graph structures can be used to identify specific genetic markers in the absence of prior knowledge. For instance, we intend to leverage our approach for the identification of transposable elements, which play important roles in eukaryotic/mammalian genomes.

Software Availability

An implementation of GraSSRep, along with the code to reproduce our results, can be found as Supplemental Code and at our GitHub repository (`https://github.com/aliaaz99/GraSSRep`).

Competing interest statement

The authors declare no competing interests.

Acknowledgment

This work was supported by the NSF under award EF-2126387. A.A., A.B., T.J.T., and S.S. conceived and designed the study. A.A. and S.S. developed the methods and theory. A.A. and A.B. performed the experiments. A.A., T.J.T., and S.S. conducted the analyses. All authors analyzed and discussed the results. A.A. and S.S. drafted the initial manuscript, which was reviewed and edited by all authors. All authors read and approved the final manuscript.

Bibliography

Agarap AF. 2018. Deep learning using rectified linear units (relu). *arXiv doi:1803.08375* .

Balaji A, Sapoval N, Seto C, Elworth RL, Fu Y, Nute MG, Savidge T, Segarra S, and Treangen TJ. 2022. KOMB: K-core based de novo characterization of copy number variation in microbiomes. *CSBJ* **20**: 3208–3222.

Breiman L. 2001. Random forests. *Machine Learning* **45**: 5–32.

Chowdhury A, Verma G, Rao C, Swami A, and Segarra S. 2021. Unfolding WMMSE using graph neural networks for efficient power allocation. *IEEE Trans SP* **20**: 6004–6017.

Čutura G, Li B, Swami A, and Segarra S. 2021. Deep demixing: Reconstructing the evolution of epidemics using graph neural networks. In *EUSIPCO*, pp. 2204–2208.

Dayarian A, Michael TP, and Sengupta AM. 2010. SOPRA: Scaffolding algorithm for paired reads via statistical optimization. *BMC Bioinformatics* **11**: 1–21.

De Boer PT, Kroese DP, Mannor S, and Rubinstein RY. 2005. A tutorial on the cross-entropy method. *Ann Oper Res* **134**: 19–67.

Gao S, Sung WK, and Nagarajan N. 2011. Opera: reconstructing optimal genomic scaffolds with high-throughput paired-end sequences. *J Comput Biol* **18**: 1681–1691.

Ghurye J and Pop M. 2016. Better identification of repeats in metagenomic scaffolding. In *WABI*, pp. 174–184. Springer.

Ghurye J, Treangen T, Fedarko M, Hervey WJ, and Pop M. 2019. MetaCarvel: linking assembly graph motifs to biological variants. *Genome Biol* **20**: 1–14.

Ghurye JS, Cepeda-Espinoza V, and Pop M. 2016. Metagenomic Assembly: Overview, Challenges and Applications. *Yale J Biol Med* **89**: 353.

Girgis HZ. 2015. Red: an intelligent, rapid, accurate tool for detecting repeats de-novo on the genomic scale. *BMC Bioinformatics* **16**: 1–19.

Glaze N, Bayer A, Jiang X, Savitz S, and Segarra S. 2023. Graph representation learning for stroke recurrence prediction. In *ICASSP*, pp. 1–5.

Hamilton W, Ying Z, and Leskovec J. 2017a. Inductive representation learning on large graphs. *NeurIPS* **30**.

Hamilton WL, Ying R, and Leskovec J. 2017b. Representation learning on graphs: Methods and applications. *arXiv doi:1709.05584* .

Haykin S. 1998. *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Iranzo J, Wolf YI, Koonin EV, and Sela I. 2019. Gene gain and loss push prokaryotes beyond the homologous recombination barrier and accelerate genome sequence divergence. *Nat Commun* **10**: 5376.

Jaiswal A, Babu AR, Zadeh MZ, Banerjee D, and Makedon F. 2020. A survey on contrastive self-supervised learning. *Technologies* **9**: 2.

Kingma DP and Ba J. 2014. Adam: A method for stochastic optimization. *arXiv doi:1412.6980* .

Kipf TN and Welling M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv doi:1609.02907* .

Kolmogorov M, Bickhart DM, Behsaz B, Gurevich A, Rayko M, Shin SB, Kuhn K, Yuan J, Polevikov E, Smith TP, et al.. 2020. metaFlye: scalable long-read metagenome assembly using repeat graphs. *Nat Methods* **17**: 1103–1110.

Koren S and Phillippy AM. 2015. One chromosome, one contig: complete microbial genomes from long-read sequencing and assembly. *COMICR* **23**: 110–120.

Koren S, Treangen TJ, and Pop M. 2011. Bambus 2: scaffolding metagenomes. *Bioinformatics* **27**: 2964–2971.

Koutrouli M, Karatzas E, Paez-Espino D, and Pavlopoulos GA. 2020. A guide to conquer the biological network era using graph theory. *Front Bioeng Biotechnol* **8**: 34.

Langmead B and Salzberg SL. 2012. Fast gapped-read alignment with Bowtie 2. *Nat Methods* **9**: 357–359.

Lapidus AL and Korobeynikov AI. 2021. Metagenomic data assembly–the way of decoding unknown microorganisms. *Front Microbiol* **12**: 613791.

Li D, Liu CM, Luo R, Sadakane K, and Lam TW. 2015. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. *Bioinformatics* **31**: 1674–1676.

Marçais G, Delcher AL, Phillippy AM, Coston R, Salzberg SL, and Zimin A. 2018. MUMmer4: A fast and versatile genome alignment system. *PLoS Comput Biol* **14**: e1005944.

Martin S, Ayling M, Patrono L, Caccamo M, Murcia P, and Leggett RM. 2023. Capturing variation in metagenomic assembly graphs with MetaCortex. *Bioinformatics* **39**: btad020.

Nurk S, Meleshko D, Korobeynikov A, and Pevzner PA. 2017. metaSPAdes: a new versatile metagenomic assembler. *Genome Res* **27**: 824–834.

Ondov BD, Starrett GJ, Sappington A, Kostic A, Koren S, Buck CB, and Phillippy AM. 2019. Mash screen: high-throughput sequence containment estimation for genome discovery. *Genome Biol* **20**: 1–13.

Salmela L, Mäkinen V, Välimäki N, Ylinen J, and Ukkonen E. 2011. Fast scaffolding with small independent mixed integer programs. *Bioinformatics* **27**: 3259–3265.

Schatz MC, Delcher AL, and Salzberg SL. 2010. Assembly of large genomes using second-generation sequencing. *Genome Res* **20**: 1165–1173.

Segarra S and Ribeiro A. 2015. Stability and continuity of centrality measures in weighted graphs. *IEEE Trans SP* **64**: 543–555.

Shakya M, Quince C, Campbell JH, Yang ZK, Schadt CW, and Podar M. 2013. Comparative metagenomic and rRNA microbial diversity characterization using archaeal and bacterial synthetic communities. *Environ Microbiol* **15**: 1882–1899.

Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, and Birol I. 2009. ABySS: a parallel assembler for short read sequence data. *Genome Res* **19**: 1117–1123.

Soucy SM, Huang J, and Gogarten JP. 2015. Horizontal gene transfer: building the web of life. *Nat Rev Genet* **16**: 472–482.

Treangen TJ, Abraham AL, Touchon M, and Rocha EP. 2009. Genesis, effects and fates of repeats in prokaryotic genomes. *FEMS Microbiol Rev* **33**: 539–571.

Treangen TJ and Salzberg SL. 2012. Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nat Rev Genet* **13**: 36–46.

Wick RR, Schultz MB, Zobel J, and Holt KE. 2015. Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics* **31**: 3350–3352.

Wooley JC, Godzik A, and Friedberg I. 2010. A primer on metagenomics. *PLoS Comput Biol* **6**: 1–13.

Wu Z, Pan S, Chen F, Long G, Zhang C, and Philip SY. 2020. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **32**: 4–24.

Yang C, Chowdhury D, Zhang Z, Cheung WK, Lu A, Bian Z, and Zhang L. 2021. A review of computational tools for generating metagenome-assembled genomes from metagenomic sequencing data. *CSBJ* **19**: 6301–6314.

Zaki N, Efimov D, and Berengueres J. 2013. Protein complex detection using interaction reliability assessment and weighted clustering coefficient. *BMC Bioinformatics* **14**: 1–9.

Zhao Z, Verma G, Rao C, Swami A, and Segarra S. 2023. Link scheduling using graph neural networks. *IEEE Trans SP* **22**: 3997–4012.

Graph-based self-supervised learning for repeat detection in metagenomic assembly

Ali Azizpour, Advait Balaji, Todd J. Treangen, et al.

Genome Res. published online July 19, 2024
Access the most recent version at doi:10.1101/gr.279136.124