COGRAM: A Computational Pipeline for Genome Assembly and Reconstruction via Optimized K-mer Sampling and De Bruijn Graph Networks

William Coggins[1] and Vijayalakshmi Ramasamay[2]

[1] Department of Mathematics, Georgia Southern University, Statesboro, GA, USA
wcoggins@georgiasouthern.edu
[2] Department of Computer Science, Georgia Southern University, Statesboro, GA, USA
vramasamay@georgiasouthern.edu

Abstract. Genome assembly and annotation accuracy fundamentally depend on the optimal selection of parameters and robust computational methodologies. In this work, we introduce COGRAM (Coggins-Ramasamy Genomic Assembly Method), a novel bioinformatics pipeline designed to enhance genome assembly and reconstruction by integrating k-mer optimization, graph theory, and ML techniques. COGRAM identifies the optimal k-mer length using KMERGENIE-inspired methods and grid search, followed by random genomic sampling. It analyzes k-mer frequency distributions and GC-content variance across genome windows, constructing a detailed de Bruijn graph. A trained graph neural network models the genomic architecture, enhancing assembly accuracy and scalability. Genome reconstruction employs cross-validation and a greedy algorithm for refinement. Tested on the Escherichia coli genome, COGRAM shows significant improvements over traditional methods, making it a powerful tool for diverse genomic assembly projects.

Keywords: KMERGENIE, grid search, COGRAM, de Bruijn, genome assembly, graph networks

1 Introduction

Genome assembly remains one of the most crucial and computationally challenging tasks in modern bioinformatics. Traditional genome assemblies rely on complex, resource-intensive methods often bound to specific operating systems (typically Linux) and require extensive parameter tuning. These constraints hinder rapid prototyping and limit accessibility for researchers comfortable in cross-platform or Windows-based environments. Recent advances in machine learning (ML) and graph-based methods have led to the development of new tools that can enhance or replace traditional assembly tools. In this work, we introduce COGRAM (Coggins-Ramasamy Genomic Assembly Method), a novel, Python-based computational pipeline that leverages optimized k-mer sampling and de

Bruijn graph networks for genome assembly and reconstruction. Unlike conventional assemblers, COGRAM is designed to be simple and user-friendly, requires no specialized assemblers like Velvet [16] or SPAdes [1], or any Linux-based structure. Our framework samples k-mers from genomic sequences and builds compact de Bruijn graphs to create efficient training datasets for graph networks. The network is then used to infer the optimal graph traversal of the true genomic sequence, effectively reconstructing the original sequence in a supervised setting.

COGRAM capitalizes on the inherent structure of genomic data by representing each k-mer as a node encoded via a one-hot vector. This representation enables the pipeline to capture complex repeat structures and sequence variations without the overhead associated with traditional assemblers. Through experimentation on model organisms such as *E. coli*, we demonstrate that even with minimal computational resources, COGRAM can achieve accurate reconstruction of genome segments. This flexible, platform-independent approach can make genome assembly more accessible to researchers and enhance the exploration of GNN methods in genomics. To summarize, COGRAM provides a streamlined and innovative framework for genome assembly, bridging the gap between conventional methods and emerging data-driven approaches. We address the following research questions, with a focus on integrating k-mer selection, graph-based learning, and multi-technique genome reconstruction approaches.

RQ1: To what extent can a graph neural network (GNN) be trained to reconstruct genomic sequences from real-world sequencing data accurately? **RQ2:** What methods are most effective for selecting optimal k-mer values for genome segmentation, and how can these be systematically parsed into edge-labeled graph structures for downstream analysis? **RQ3:** How does integrating multiple techniques, such as k-mer optimization, graph construction, and neural learning, impact the overall accuracy and scalability of genome reconstruction pipelines?

2 Related Work

K-mer Optimization & Sampling: K-mer selection critically impacts assembly quality. Tools like KMERGENIE [4] estimate the optimal k based on k-mer abundance histograms. GenomeScope [14] uses k-mer frequencies to model genome size and heterozygosity, and has succeeded on bacterial and eukaryotic genomes, although often with static assumptions. COGRAM improves on this by combining histogram-based methods with random contig sampling, aligning with stratified data selection methods [7]. **Graph Construction & Representation:** Using de Bruijn graphs (DBGs) for genome assembly is well-established (e.g., Velvet [16] and SPAdes [1]). COGRAM innovates by collapsing non-branching unitigs and labeling edges based on read order. This aligns with advances in succinct DBG representations, such as Bcalm2 [3], and space-efficient edge labeling methods. **Graph Neural Network Training:** COGRAM's use of GCNConv + edge-MLP aligns with graph-based models in bioinformatics. GraphAligner [11] and DeepGraphPose [10] applied similar GNN structures for biological sequence understanding. Training with edge scoring via sigmoid + early stopping mirrors strategies from GAT and GCN++ architectures. **Reconstruction and Evaluation:** The hybrid greedy + beam search method in

COGRAM improves the classical greedy walk used in OLC assemblers. Hybrid reconstruction draws inspiration from hybrid de novo assembly (e.g., Unicycler [15]) and contig scaffolding with constraint satisfaction (e.g., SSPACE [2]).

Selected genome assembly research and their methodological approaches, and a comparison of traditional and ML-based genome assembly pipelines (Table 1 reveal that integrating Simunovic's GNN edge classifier with Vrcek's GCN path predictor into a unified pipeline for graph simplification and path reconstruction would be powerful. A dynamic k-mer selection will optimize k based on local graph structure. Cross-platform generalization enhances GNN and Transformer models using diverse error profiles for better reconstruction.

Table 1: Key methodological features: traditional vs. ML-based genome assembly

Feature	**Traditional(Cha [3], Duan [6])**	**ML-Based (Simunović [12], Vrček [13], Luo [9])**
k-mer Optim.	Manual or modeled (KmerGenie)	Fixed k, not the focus
Error Detection	Heuristic thresholds	GNN-based classification
Path Selection	Eulerian traversal, greedy heuristics	Learned path via GCN/Transformer
Evaluation	N50,CEGMA,GC-coverage plots	NGA50, precision/recall on edges
Data Type	Short reads (Illumina)	Mixed; focus long reads (HiFi)

3 Overview of COGRAM Framework

This section presents the COGRAM framework components (Fig. 1).

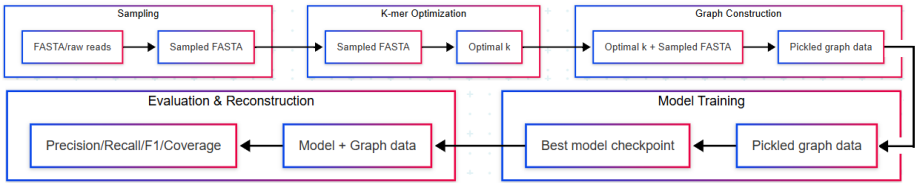


Fig. 1: COGRAM Framework

3.1 Optimization of K-value Selection and Sampling Process

We have constructed a new, naive optimizer that scans over a range of k-values, counts the k-mer abundances, filters out singletons, and picks the k with maximal "genomic" k-mer count (abundance $\geq$ threshold). The text-based FASTA format represents nucleotide or amino acid (protein) sequences represented using single-letter codes [8].

For each chromosome in the input FASTA file, we randomly sample one or more contiguous windows of a specified length (say, 10000) instead of processing entire chromosomes, which can significantly slow down the development and testing of the pipeline. By using smaller "mini" chromosomes, we can quickly iterate through the pipeline for testing purposes.

3.2 Data Processing and Exploratory Data Analysis

This step transforms the sampled FASTA data into two objects:(i) **raw** k-mer overlap graph for network training, and (ii) **unitig-compressed** graph for faster assembly. **contig** is a continuous stretch of DNA sequence reconstructed by an assembler, which is arrived at when an assembler stitches together reads to

build as long a sequence as possible without ambiguities or unresolved repeated patterns. By contrast, an **unitig** is a maximal non-branching path in a de Bruijn graph, collapsed into a single node using the following steps: (i) Build a DBG whose nodes are k-mers and edges connect overlapping k-mers. (ii) Wherever a node has exactly one edge in and one edge out (i.e., no branching), merge them. (iii) Repeat this until every remaining node has either >1 in-edge or >1 out-edge. Unitigs are unambiguous paths to simplify graph reconstruction, allowing quicker traversal of large sections. They maintain granularity for model training and significantly reduce the graph size by merging linear chains into one node.

This segment serves to inspect our data and ensure that what we are working with is of acceptable quality. We examine two main features: (i) the k-mer frequency abundance and (ii) the GC content window. For each record, we compute the length, A/C/G/T counts, and overall GC content. We then compute the k-mer frequency and plot it on a log scale (because we are on the order of hundreds of thousands). Then, we compute the GC sliding window to visualize the percentage of the record that contains either guanine or cytosine. This content is well-studied and provides us with insight into the stability of the genome. Aside from genome stability, good data analysis allows us to identify uncharacteristic behaviors such as wild GC fluctuations and contamination in the genome, and to be comfortable with our input data.

3.3 GCN Training and Assembly Analysis

The model's training helps predict which edges in the raw K-mer graph correspond to a true sequence or ground truth cycle, like an **edge classifier**. This is the heart of our ML approach - teaching a graph neural network to recognize real adjacencies versus spurious overlaps in the graph. Once the graph is loaded in, the graph data is split into a training, validation, and testing set with a 70/15/15 split. The model clearly defines an edge classifier: two GCN layers to embed each node, then an MLP that takes a pair of node embeddings and outputs a score for the edge. It then runs a standard deep-learning training model with BCE loss, Adam optimizer, learning-rate scheduling, and early stopping, and saves the best model.

Our model uses a hybrid reconstruction technique to traverse the unitig graph and the best model checkpoint and reconstruct the original sequence as best as possible. The assembly validation step measures real-world performance on both a global scale and on a chromosome-by-chromosome basis. We use the following three major steps: **(i)** tune an edge-score threshold for the best validation F1, **(ii)** reconstruct the Hamiltonian cycle with a greedy/beam hybrid, and **(iii)** report precision, recall, F1, and coverage globally and per "chromosome".

We use the term "chromosome" lazily here, as it refers to a sample region, not an actual biological chromosome. In this two-stage reconstruction, the greedy walk takes the highest-scoring edges until we recover a user-defined fraction of the nodes. After the greedy walk, we switch to a beam search at branch points, saving top candidates and gluing in the rest of the cycle. The primary strengths of COGRAM are extensible modular architecture, data-driven sampling, and integration of graph-based learning for structure-aware modeling.

4 Experimental Setup

Clean and ready for work (FASTA data can be downloaded directly), datasets are gathered from the NIH genome database cite nihgenomelibrary. For initial testing, we chose *E. coli*, a short genome that only contains one biological chromosome ($\approx$ 4 Mb) for manageable sampled graphs. The two hyperparameters of the model architecture are: **in channels** sets the dimensionality of the input features. In our case, we choose 4 x k for one-hot encoded k-mers. For example, A is encoded as [1, 0, 0, 0]. The **hidden channels** are set to 32, which is the size of each GCN embedding layer. As the hidden layers are adjusted, we can monitor various F1 values during validation. The edge multi-layer perceptron (MLP) has two linear layers: First layer maps $2 * \text{hidden} \rightarrow \text{hidden}$ for concatenated source and target embeddings; Second layer maps $\text{hidden} \rightarrow 1$ for edge logits. With a maximum of 100 training epochs (early stopping), training ended after 70 epochs, as the validation loss failed to improve for 10 consecutive epochs. If validation loss plateaus early, we adjust the Adam optimizer's learning rate, and a scheduler halves the learning rate if validation loss doesn't improve for 5 consecutive epochs.

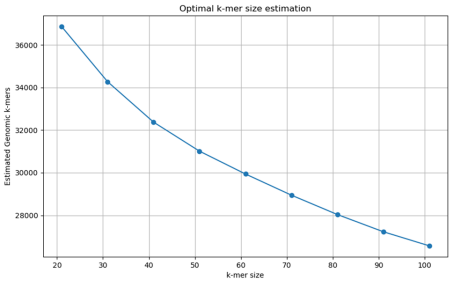
The GNN functions using a typical message passing recipe: Message MLP, Sum Aggregator, Update MLP, Edge Classification, Greedy Decoder. The message MLP combines neighbors' state and edge features, and can be inspected by looking at learned weight magnitudes. The sum aggregator aggregates the raw messages and can be inspected by comparing with the mean: sum preserves the coverage signal. The updated MLP combines the old state and new aggregates, allowing for inspection by tracking gradients to assess the importance of each feature. Edge-classification scores edges for inclusions in the final cycle, and can be inspected for precision-recall per feature set. And finally, the greedy decoder stitches together unitigs into a circular genome.

For reconstruction, we control three parameters: the greedy fraction, beam width, and top-scoring edges during expansion. A larger greedy search fraction speeds up the process, but may sacrifice global exploration. If it's too low, we risk entering beam search too early, which increases the computational load. We set the beam width = 3 to evaluate three candidate paths and retain the top 5 highest-scoring outgoing edges at each iteration of expansion. This *hybrid algorithm* accelerates reconstruction, enabling significant genome exploration.

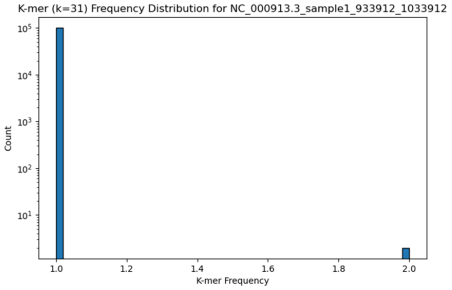
To fine-tune training, sweep over several greedy fractions in the interval of [0.1, 0.3, 0.5]. For each sweep of the greedy fractions, we also sweep over beam width and top k values: (3, 5), (5, 10), (10, 20). During these sweeps, we plot the F1 score and runtime for each configuration, and then we select a preferred regime. We used an NVIDIA GeForce RTX 3070 [5] and it lasted approximately 15 minutes. We achieved fast training times due to the small size of our graph for this genome. Larger genomes will require more extensive model fine-tuning, as noted in the shortcomings section.

5 Preliminary Results and Discussion

The k-mer distribution of a sample is shown (Fig. 2a), and we choose k = 21. The large peak at 1.0 indicates that most k-mer segments are unique. The small peak at 2.0 means there are some duplicates, but very few by comparison.



(a) Naive K-mer Optimization



(b) K-Mer Frequency Abundance for Sample S1

The GC-content window for sample S1 is relatively uniform with only minor deviations from 50%, and it is assumed standard for any genome (Fig. 3). **Reconstruction and Validation:** After training and reconstructing the genome using our hybrid algorithm, Precision = 0.967, Recall = 0.947, F1 = 0.957, Coverage = 0.979, indicate high confidence that reconstructed adjacencies are real (precision), high ability to capture most true adjacencies in the genome (recall), an excellent harmonic tradeoff (F1 Score), and almost all genome regions (nodes) were included in the reconstructed cycle at least once (Coverage). The per-chromosome (sampled regions) scores are shown in Table 2. A lower 'visited

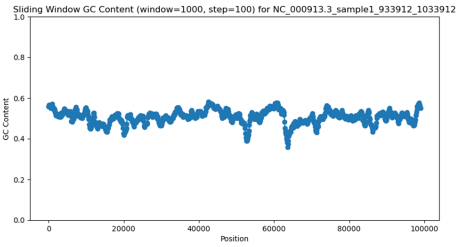


Fig. 3: Sliding GC Content Window(S1)

Table 2: Per-sample reconstruction performance. * = 'visited nodes/Total nodes'

Sample ID	Greedy*	Beam Behavior	Prec.	Recall	F1	Coverage
`Sample S1`	23/79	No expansions, stop	0.678	0.513	0.584	0.759
`Sample S2`	1193/3977	Iter. 1 000&2 000 then stop	0.986	0.961	0.973	0.975
`Sample S3`	68/227	No expansions, stop	0.929	0.580	0.714	0.626
`Sample S4`	762/2543	No expansions, stop	0.967	0.535	0.689	0.553
`Sample S5`	9/31	Greedy only	1.000	1.000	1.000	1.000

nodes/Total nodes' (Vis/Tot) ratio suggests that the greedy path terminated early, possibly missing many parts of the graph. A higher Vis/Tot ratio (S2: 1193/3977) indicates that the algorithm covered more of the graph, although not necessarily effectively. This measure of search coverage indicates the extent to which the graph was explored during greedy decoding.

The key observations and inferences are: (i) Beam search (S2) achieves the highest performance (F1 = 0.973, Coverage = 0.975), indicating its effectiveness in complex graphs. It significantly improves recall and coverage, especially for larger graphs. (ii) Greedy-only methods (S1, S3, S4) yield lower recall and F1,

except for S5, which achieves perfect scores likely due to low complexity. This reconstruction can be sufficient for small, simple graphs (e.g., S5). (iii) High node visits (e.g., S4: 762/2543) do not guarantee better performance—greedy can still underperform. The performance depends more on search strategy and graph structure than on the number of nodes visited.
Related Application Domains include: **Error Detection in Genome Sequencing for Disease Diagnostics:** Reframing the problem to identify anomalous regions in the de Bruijn graph helps classify sequencing errors and variant breakpoints. Training on clean data and simulated mutations enables the GNN to flag suspicious edges, enhancing personalized medicine workflows, particularly for long-read or single-cell sequencing, where realigning flagged regions facilitates variant discovery. **Social Network Analysis:** The proposed framework can be used for social graphs (nodes: users, edges: interactions). Features, demographics and activity patterns (instead of k-mer encodings) enable GNN to predict latent ties, future links, detect bot activity, and fraudulent accounts. Applications include recommender systems, misinformation tracking, and fraud detection.

6 Discussion and Threats to Validity

The overall time complexity of the pipeline is $O((E \cdot H) + E + N) \approx O(N)$, scaling polynomially with the dataset size. (E: number of edges, N: number of nodes in the DBG, and H:hidden dimensionality of the GNN. The F1 score and global genome coverage are high, but results vary for sampled regions. Large samples (S2), enable greedy searches to cover most nodes, resulting in a high F1 score. Medium samples (S3, S4), can trap the beam search, limiting recall and coverage. Small samples often yield accurate results due to their trivial nature and small search space. The current threat to our validity is the naivety of k-mer optimization and the minimal training. The model is tuned for only a single sample, which should be improved and will be expanded on. The pipeline will also be reworked to allow for the use of FASTQ data.

7 Conclusion and Future Work

This compact test model demonstrates strong potential for accurate genome reconstruction and effective training on limited datasets. In the near term, we aim to refine model parameters and explore a broader range of configurations (see Section III). Long-term, our goal is to develop a fully end-to-end *de novo* assembly pipeline and benchmark it against established assemblers. The current k-mer optimization is simplistic; a more principled approach—such as sweeping over k values starting at $k = 21$—may yield more consistent results across diverse genomes. As noted by Chikhi and Medvedev [4], adopting a two-component mixture model on k-mer abundance histograms can better distinguish error-derived from genomic k-mers and help select an optimal k. Future work will also incorporate N50 and L50 evaluations. Our current Hamiltonian cycle approach to genome reconstruction is computationally challenging; beam search serves only as a heuristic with no guarantees. We do not yet implement graph cleaning or error correction, unlike mature assemblers that prune low-coverage paths and resolve bubbles. While this is not an issue for clean FASTA data, real reads (e.g.,

FASTQ) will require robust graph cleaning. Transitioning to an Eulerian-based assembly with graph simplification may offer greater robustness and scalability.

References

[1] Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., Pyshkin, A. V., Sirotkin, A. V., Vyahhi, N., Tesler, G., Alekseyev, M. A., and Pevzner, P. A. (2012). Spades: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477.

[2] Boetzer, M. and Pirovano, W. (2011). SSPACE: scaffolding pre-assembled contigs using paired-read data. *Bioinformatics*, 27(4):578–579.

[3] Cha, S. and Bird, D. M. (2016). Optimizing k-mer size using a variant grid search to enhance de novo genome assembly. *Bioinformation*, 12(2):36–40.

[4] Chikhi, R. and Medvedev, P. (2014). Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37.

[5] Corporation, N. (2020). Geforce rtx 3070 graphics card. Accessed: 2025-05-03.

[6] Duan, Y., Li, Y., Zhang, J., Song, Y., Jiang, Y., Tong, X., Bi, Y., Wang, S., and Wang, S. (2023). Genome survey and chromosome-level draft genome assembly of *Glycine max* var. dongfudou 3: Insights into genome characteristics and protein deficiencies. *Plants*, 12(16):2994.

[7] Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., and McVean, G. (2012). De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature Genetics*, 44(2):226–232.

[8] Lipman, D. J. and Pearson, W. R. (1985). Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441.

[9] Luo, J., Fang, H., Xie, L., Wang, Y., and Gao, X. (2024). Gtasm: A genome assembly method using graph transformers and hifi reads. *Frontiers in Genetics*, 15:1495657.

[10] Mathis, A., Mamidanna, P., Cury, K. M., Abe, T., Murthy, V. N., Mathis, M. W., and Bethge, M. (2018). Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21(9):1281–1289.

[11] Rautiainen, M. and Marschall, T. (2020). Graphaligner: Rapid and versatile sequence-to-graph alignment. *Genome Biology*, 21(1):253.

[12] Simunovic, M. et al. (2023). Graph neural network meets de bruijn genome assembly. In *Proceedings of ISPA 2023 Conference*.

[13] Vrcek, L. et al. (2022). Learning to untangle genome assembly with graph convolutional networks.

[14] Vurture, G. W., Sedlazeck, F. J., Nattestad, M., Underwood, C. J., Fang, H., Gurtowski, J., and Schatz, M. C. (2017). Genomescope: fast reference-free genome profiling from short reads. *Bioinformatics*, 33(14):2202–2204.

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

[16] Zerbino, D. R. and Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–829.