Reconstruction of short genomic sequences with graph convolutional networks

Lovro Vrček*†, Xavier Bresson‡, Thomas Laurent§, Martin Schmitz†‡, Mile Šikić*†
* Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia
† Genome Institue of Singapore, A*STAR
‡ National University of Singapore
§ Loyola Marymount University
lovro.vrcek@fer.hr, mile.sikic@fer.hr

***Abstract*—Genome reconstruction, without prior knowledge about the sequence we are reconstructing, is performed with tools called *de novo* genome assemblers. These tools rely on numerous heuristics and usually provide a fragmented reconstruction, even for sequences shorter than the entire genomes or chromosomes. One of the most common approaches to *de novo* assembly, called Overlap-Layout-Consensus, constructs a graph from short overlapping fragments, which heuristics then simplify and find a path through. In this work, we explore how graph neural networks (GNNs) can assist with this task, and show that the GNN-based Layout phase can reconstruct longer sequences than naive search algorithms or heuristics deployed in de novo assemblers, with no significant difference in compute time on sequences up to 10 Mbp in length.**

***Keywords*—genome assembly, graph neural networks**

I. INTRODUCTION

Fast and accurate *de novo* genome assembly is one of the most difficult problems in bioinformatics and it remains unsolved to this day. It focuses on reconstructing the original genomic sequence from a sample of shorter overlapping fragments, called reads, without any prior knowledge about the original sequence. The first major achievement of *de novo* genome assembly happened in the early 2000s when the Human Genome Project was finished, an effort that took over a decade and cost billions of dollars [1]. The reported results of the project were that 99% of the genome had been reconstructed with less than 400 gaps. Unfortunately, that was not entirely correct, as only the euchromatic portion of the genome was considered while the heterochromatin was left out. When the heterochromatin regions—which include centromeres, telomeres, and tandem gene arrays—are also taken into account, the final result is that more than 5% of the entire genome was either missing or incorrect.

Since then, the sequencing technologies have improved significantly in many ways, but most notably in terms of the lengths and accuracies of the reads they produce. At the forefront of the latest sequencing technologies are the HiFi reads developed by PacBio [2] and the ultra-long reads by Oxford Nanopore Technologies [3], both of which were crucial for the most recent breakthrough in the field of de novo genome assembly—full reconstruction of the entire human genome, with no regions left unsolved [4]. Nevertheless, this was enabled not only by the latest sequencing technologies, but also by a tremendous effort of numerous researchers and bioinformaticians who used various *de novo* assembly tools and manually inspected large genomic regions.

One of the more common approaches to *de novo* genome assembly, which was also the one used in the recent reconstruction of the human genome, is the Overlap-Layout-Consensus (OLC) paradigm. In the Overlap phase, the reads in the sample are mapped onto each other in an all-versus-all manner in order to find overlaps between them. All reads that are entirely contained in other reads are removed from further processing. From the rest of the overlapped reads, an assembly graph is built—a graph in which nodes represent reads and edges represent the suffix-prefix overlaps between the reads. In the Layout phase, the assembly graph is simplified in order to find a path through it that would reconstruct the original genome. Finally, in the Consensus phase, all the reads are aligned to the reference in order to clean the assembly sequence of errors that happened during the rest of the process.

In the ideal scenario, the Layout phase would be formulated as finding a Hamiltonian path over the assembly graph—visit every node in the graph exactly once. However, due to imperfect read qualities, sequencing artifacts, and repetitive genomic regions, finding a Hamiltonian path is usually not possible and an unknown number of nodes and edges has to be removed. Because of this, instead of finding a path through the graph directly, modern assemblers rely on heuristics to simplify the entire graph, by iteratively removing nodes and edges deemed unnecessary, such as removing transitive edges, trimming tips, and popping bubbles [5], [6]. Frequently, however, assembly graphs are highly complex, and even this is not a straightforward task, as some regions cannot be simplified by the current heuristics. One such region can be seen in Figure 1 on the left, where the blue nodes are those that should be traversed for the ideal assembly, while the orange nodes should be avoided. For lack of a better method, these complex regions are cut out of the graph, and we are left with multiple fragments of the original genome instead of the continuous sequence. To this day, the problem of fragmentation continues to plague all the existing assemblers.

The central part of reconstructing the full human genome was untangling these complex regions in the assembly graph instead of cutting them. This approach required an abundance of data, various tools, and a detailed manual inspection of certain regions and individual reads. Reproducing such efforts on new genomes seems infeasible, and thus fast and accurate de novo genome assembly remains elusive.

In this work, we propose a novel approach to de novo genome assembly, one based on deep learning and finding a path through the graph instead of relying on handcrafted heuristics to simplify the graph. We train a non-autoregressive model based on gated graph convolutional network (GatedGCN) introduced by [7] that takes an assembly graph and outputs a score for each edge. These scores can then be used to guide a search algorithm over the graph, producing a path that represents the reconstructed genome. Our idea is illustrated in Figure 1 on the right—a depiction of the graph traversal done during the training, where in each step we calculate the probabilities for traversing each neighbor of the current node. For this, we create a synthetic dataset of reads from the real human genomic data, and generate assembly graphs prior to any simplification steps in order to avoid any errors such simplifications might produce. We believe this approach could reduce the fragmentation of the reconstructed genomes, especially in the entangled regions.

Lately, graph neural networks (GNNs) have been applied to a variety of biological problems, ranging from drug design [9] and protein interactions [10], to predicting anticancer foods [11]. However, to the best of our knowledge, the only work based on geometric deep learning that addresses *de novo* genome assembly is [12], which only simulates the deterministic simplification algorithms in the Layout phase, instead of tackling the tangled regions themselves. Moreover, the graphs used in [12] were completely synthetic, not generated from the biological sequences. Here, we go a step further by starting with the complete human genome and focusing on one of the main problems in genome assembly.

The rest of the paper is composed as follows: In Section II we describe the dataset preparation, in Section III we describe the architecture of the used model, in Section IV we state the performed experiments, and in Section V we discuss the results. Finally, Section VI concludes this paper.

II. DATASET

Generating the dataset can roughly be separated into three tasks. First, we simulate the genomic data from the human genome. Second, we adapt an existing tool for *de novo* genome assembly, Raven [6], to build an assembly graph for each set of reads. Finally, we implement an algorithm that leverages the information stored during the read-simulation phase and finds a ground-truth path through an assembly graph. We use these paths as the supervision signal during the training phase.

A. Simulating reads

To simulate the reads, we start with the recently reconstructed human genome called CHM13 [4], which consists of one chromosome from each of 23 pairs of human chromosomes. In the assembly process, each chromosome would ideally be represented by a single component of the large disconnected graph of the entire human genome, but in a realistic case, several chromosomes could be connected into a single component. Starting from a simpler scenario, we isolated one chromosome and split it into shorter "mini-references", each 2 million base pairs (bp) long.

During the sequencing process of the real samples, all information about the ordering of the reads and their position on the genome is lost. This is the main reason why we are not using sequenced human data, but simulate our own by sampling the reads from the mini-references. Thus, we are able to store the positional information for each read, which allows us to construct the ground-truth path for training.

The reads are simulated by mimicking the sequencing process. The mini-reference is sampled in a manner that each base of the mini-reference is covered approximately 20 times, which represents the number of times genomes are copied and fragmented prior to sequencing (sequencing depth). The lengths of the reads are sampled from a normal distribution with a mean value of 20,000 and a standard deviation of 1,500. These values were determined empirically so that the simulated reads would resemble the real PacBio HiFi reads. In contrast to sequencers, we introduce no errors to the simulated data to facilitate. Considering that the average accuracies of the real PacBio reads are usually over 99% and that there are tools that perform error correction of reads [13], we believe that this gives a decent approximation of the real data.

B. Generating graphs

Once the reads are simulated, we can construct the graphs. For this purpose, we use an adapted version of Raven, an assembler that follows the OLC paradigm and can output the assembly graphs at different stages of the Layout phase [6]. Additionally, we required Raven to keep only the perfect overlaps between the reads. Therefore, unless two reads have a suffix and a prefix which are matching in all the bases, they will not be connected with an edge in the assembly graph.

At the end of the Overlap phase, the assembly graph is constructed. We output the generated graph at the start of the Layout phase, prior to any simplification algorithm applied, in order to avoid errors that can occur during the simplification steps. The end result of this entire process is 50 graphs, each containing around 3,500 nodes and 50,000 edges. Considering that the reads are perfect and the similarity score of each overlap is 1.0, we only rely on overlap lengths as edge features and use no node features (or rather, we specify the feature of each node to be 1). A possible approach would be to encode the

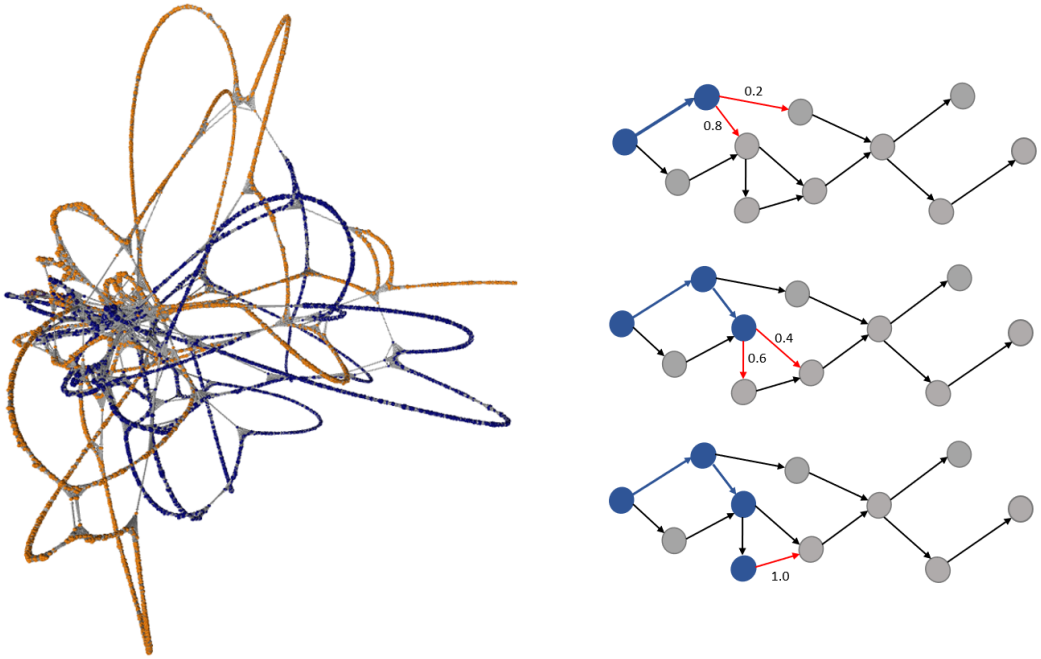

Fig. 1: **Left**: An assembly graph of one part of chromosome 11, containing the repetitive region. Blue nodes should be traversed, while the orange ones should be avoided. Figure generated with Graphia [8]. **Right**: Three steps of traversing the graph during the training where, in each step, the probabilities are computed for traversing all nodes neighboring the current node.

genomic sequences with a 1D-CNN and use the read-sequence encodings as node features and overlap-sequence encodings as additional edge features, but we leave that for future work. Therefore, in order to train on the generated graphs, the last thing needed is the supervision signal—a ground-truth path for each graph.

C. Ground-truth paths

A ground-truth path is a path in the assembly graph that produces the longest length of the reconstructed genome. For this, we utilize the positional information that was stored for each read during the sampling process and implement an algorithm resembling depth-first search, with an additional preference for successor nodes that are closer to the current node on the mini-reference. Neighbors that don't share a position on the mini-reference with the current node are avoided. Although the reads and the overlaps in the created graphs are both perfect, connections between the distant genomic regions can still exist. The main culprits for this are repetitive regions, regularly found in telomeres, centromeres, and highly duplicated ribosomal RNA genes.

In a way, this algorithm can be described as an exhaustive search with an oracle. Even though the oracle—the reads' positions on the mini-reference—guarantees that the reconstructed genome will be optimal, we noticed cases when the length of the reconstruction was less than that of the original mini-reference. There are two reasons for this. First, during the Overlap phase, all the reads are trimmed, which slightly reduces them in length. Since the ends of the mini-reference are covered only by the ends of the reads, trimming the reads necessarily leads to a loss of information. Second, and the more interesting case, happens due to an error during the Overlap phase resulting in a fragmented assembly graph. This occurs even though the sampled reads can cover the entire mini-reference (apart from a few bases on either end). We believe this happens due to a combination of repetitive regions and the discarding of reads completely contained inside the other reads. However, more analysis is needed to make a definite claim.

Notice that here we find just a single path through the graph, whereas in reality, multiple paths could lead to the optimal solution. This is particularly true in the case of diploid and polyploid genomes, but even in the case of haploid genomes, such as CHM13, there are transitive edges that do not influence the final assembly traversed. However, for the sake of an easier problem statement, we focus only on the best neighbor for each node.

III. MODEL ARCHITECTURE

The proposed model for obtaining the probability scores is non-autoregressive and can be split into three parts—encoder, processor, and decoder.

A. Encoder

A layer that transforms node features $x_i \in \mathbb{R}^{d_v}$ and edge features $z_{ij} \in \mathbb{R}^{d_e}$ into the d-dimensional node and edge representations. As stated in Section II, we use $x_i = 1.0$

and $z_{ij} = \text{len}(i \to j)$, thus making $d_v = d_e = 1$. Encoder for both node and edge features is a single linear layer:

$$h_i^0 = W_1 x_i + b_1 \in \mathbb{R}^d, \tag{1}$$

$$e_{ij}^0 = W_2 z_{ij} + b_2 \in \mathbb{R}^d, \tag{2}$$

where h_i^0 is the initial node representation of the node i, e_{ij}^0 is the initial representation of the edge $i \to j$, and $W_1 \in \mathbb{R}^{d \times d_v}, W_2 \in \mathbb{R}^{d \times d_e}$, $b_1, b_2 \in \mathbb{R}^d$ are learnable parameters.

B. Processor

The main part of the network consists of multiple GNN layers. For this task, we modify the GatedGCN [7] by including the edge representations and a dense attention map η_{ij} for the edge gates, as proposed in [14], [15].

The motivation for using this layer comes mainly from its performance on the Traveling Salesman Problem (TSP) [15]. Since the problem of finding the optimal walk on the assembly graph is similar to that of TSP, it makes sense to reuse the architecture. In addition, it was shown that the GatedGCN outperforms many other models in several tasks [16], which gives us further reason to use this architecture.

Let now node i be the node whose representation we want to update, and let its representation after a layer l be h_i^l. Also, let all the neighbors of node i be denoted with j. Then, in the layer $l+1$, the node and edge representations will be:

$$h_i^{l+1} = h_i^l + \text{ReLU}\left(\text{BN}\left(A^l h_i^l + \sum_{j \in N(i)} \eta_{ij}^{l+1} \odot B^l h_j^l\right)\right),$$

where all $A, B \in \mathbb{R}^{d \times d}$ are learnable parameters, ReLU stands for rectified linear unit, BN for batch normalization, and $\odot$ for Hadamard product. The edge gates are defined as:

$$\eta_{ij}^l = \frac{\sigma\left(e_{ij}^l\right)}{\sum_{j' \in N(i)} \sigma\left(e_{ij'}^l\right) + \epsilon} \in \mathbb{R}_+^d \tag{3}$$

where σ is the sigmoid function, and ϵ is a small value in order to avoid division by zero.

C. Decoder

A multi-layer perceptron (MLP) decodes the obtained representations into the probability scores. Probability score p_{ij} for traversing an edge $i \to j$ is computed from node representation of nodes i and j, as well as edge representations of the edge $i \to j$, all after the final GatedGCN layer L:

$$p_{ik} = \text{MLP}(h_i^L \parallel h_k^L \parallel e_{ij}^L), \tag{4}$$

where $(\cdot \parallel \cdot)$ is the concatenation operator.

IV. EXPERIMENTS

A. Training

Processing of a graph during one training epoch is done iteratively. In a single iteration, the graph is fed to the model providing us with scores for all the edges. We start the traversal from the starting node of the ground-truth path and take w steps, where w is a hyperparameter and we refer to it as the walk length. In each step, the best successor is predicted from their probability scores, while the correct next node is obtained from the ground-truth path. The cross-entropy loss is computed over the current node's successors, followed by teacher forcing where we choose the successor given by the ground-truth as the next node. This is repeated for w steps, after which the computed losses are averaged, the backpropagation is performed, and the next iteration starts, continuing from the last visited node. This is repeated until the end of the ground-truth walk. It is important to notice that, while computing the loss, the number of candidates in each step can vary, as the number of successors also varies.

The training was performed on a dataset consisting of 50 graphs with a 30/10/10 train/validation/test split. Each graph had about 3,500 nodes and 50,000 edges, all of them generated from 2 Mbp mini-references coming from chromosome 11. We used Adam optimizer [17], with the initial learning rate of 10^{-4}. We also decay the learning rate multiplying by with 0.9 in case there was no improvement in the validation loss for 5 epochs. The evaluation metric used during training to keep track of the learning process was the accuracy of predicting the best next neighbor, which we also refer to as the local reconstruction metric. The entire training was done on a single Nvidia A100 GPU.

Predicting the best successor is the only metric used during the training, since it is easy to calculate the corresponding loss and accuracy. In a way, the better the model follows the ground-truth, the better it should reconstruct the entire sequence. However, there are some situations where choosing the incorrect node makes no difference (e.g., transitive edges such as $i \to k$, when there also exist $i \to j$ and $j \to k$), as well as situations where choosing the correct node is crucial (e.g., dead-end nodes). Therefore, even though used for training, this is not the best metric for genome assembly in general. Still, we expect a certain transferability between the local and the global task, and this approach simplifies the training significantly. In addition, we also perform teacher forcing during the validation and testing—even though the model makes a mistake, we will put it back on the right path for the sake of easier evaluation of the training process.

B. Inference

At inference, the predictions are not performed iteratively, but all in one go—we feed the graph to the model and find a path with the greedy algorithm following the probability scores. The greedy algorithm runs until it

reaches a node without outgoing edges or all of the node's successors have already been visited. Since there is no good option for choosing the starting node, we run the greedy search from all the nodes that have in-degree zero and choose the longest walk. At inference, we cannot evaluate the performance by the accuracy of choosing the best next neighbor, since there is no ground-truth. Moreover, *de novo* genome assemblers usually work in a completely different way, so this would make the comparison against them impossible. Therefore, we evaluate our model on two other tasks—length of the reconstruction and the execution time:

a) Reconstructed sequence length.: We measure the reconstructed length as the percentage of the original mini-reference length from which the reads were sampled. This metric is robust to errors while choosing transitive edges, but harshly punishes choosing dead-end nodes and wrong paths in general. Ideally, the length ratio would be 1, but it is expected that it will be slightly less due to the way the reads are simulated and processed prior to the Layout phase (e.g., trimming of reads mentioned in Section II-C).

b) Execution time.: One of the main pitfalls for many assemblers is their execution time. We evaluate the time needed for the model to process the graph and the search algorithm to find the best path through it. All the experiments related to execution time were performed on a single Intel Xeon E5-2698 v4 CPU.

We benchmark our model on the mentioned tasks against three other approaches. First, the naive approach where the scores are calculated as normalized overlap lengths. This approach comes down to running a greedy algorithm over the overlap lengths, and thus we refer to it as the greedy in the next section. The second approach is the one used for obtaining the ground-truth paths as explained in Section II-C, the exhaustive search with oracle. Finally, we also compare the developed model against an existing assembler Raven, which was used to generate the assembly graphs in Section II-B.

V. RESULTS

The results reported here were obtained by the best-performing model, which had 4 GatedGCN layers, latent dimension 32, and the walk length was 10. The MLP classifier consisted of a single layer. During training this model achieved a 99.90% accuracy on the test set and upon deeper inspection, we noticed that the errors mostly consisted of traversing transitive edges.

Table I shows the evaluation results of our model (GNN) benchmarked against greedy approach, Raven, and exhaustive search with oracle (ES*). For the reported lengths, first the percentage of the reconstruction was calculated for each graph, after which all the percentages were averaged. With the aim to test how our model scales to larger graphs, and thus larger genomes, we trained it on graphs generated from 2 Mbp mini-references, and evaluate on graphs generated from 2, 5, and 10 Mbp mini-references. The number of nodes and edges scaled roughly linearly with the length of the mini-reference. For the evaluation on graphs generated from 2 Mbp mini-reference, we reuse the graphs from the test set, coming from chromosome 11. Additionally, we cut chromosome 10 into 5 Mbp mini-references and chromosome 12 into 10 Mbp mini-references. There was no particular reason for choosing the chromosomes 10, 11, and 12. All three chromosomes are of similar complexity, so the extracted mini-references should also be similar, with length being the main difference between them.

a) GNN vs ES*.: The developed GNN-based method doesn't manage to reconstruct the genomes as well as the exhaustive search, but does not fall far behind either. The difference comes from the cases where our model got stuck in a dead-end node, similarly to what happened to the greedy algorithm.

b) GNN vs Greedy.: We notice that our model consistently outperforms the greedy algorithm in terms of reconstructed length. This means that the model managed to leverage the graph topology in order to avoid some pitfalls where the greedy algorithm got stuck, such as dead-end nodes. The greedy algorithm is slightly faster than our model, which comes as no surprise. Interestingly, the difference in execution times is relatively small, mainly because the neural network we used is also relatively small, and so are the graphs, so a single forward pass through it can be efficiently done on a single CPU.

c) GNN vs Raven.: We notice that Raven slightly outperforms the GNN model on the two larger datasets, but on the 2 Mbp dataset falls behind even the greedy approach. After a more thorough analysis, we noticed that Raven underperformed on only one graph, while on all the other graphs in that dataset, it was as good as the GNN. The graph on which Raven failed came from a highly repetitive region, and since Raven tries to simplify the graph by removing nodes and edges, it ends up cutting the graph into numerous fragments, the longest one being only around 15% of the original mini-reference. GNN managed to correctly find a path through that graph. This is a critical result, as the repetitive regions are the main reason why the assemblers fail to accurately reconstruct genomes. At the same time, Raven is faster than the GNN, which is partially due to more efficient implementation of the algorithms performed. However, with differences being only up to a few seconds, this is insignificant given that the bottleneck of the *de novo* assembly is usually the Overlap phase.

Overall, we show that the model performs well consistently over graphs of different sizes and even different chromosomes, both in terms of accuracy and speed. Yet, human chromosomes are up to 250 Mbp long, so a definite conclusion on how it would perform in such a setting cannot be made.

VI. CONCLUSION

In this work, we introduce a novel approach to solving *de novo* genome assembly based on graph neural networks and finding a path through the assembly graph. We created

TABLE I: Comparison of GNN, naive approach, Raven, and exhaustive search.

Method	2 Mbp		5 Mbp		10 Mbp	
	length [%]	time [s]	length [%]	time [s]	length [%]	time [s]
ES*	99.79 ± 0.16	2.5 ± 0.1	95.82 ± 14.67	7.5 ± 1.1	95.93 ± 13.94	23.0 ± 5.0
Greedy	93.50 ± 17.88	0.2 ± 0.0	90.21 ± 24.01	0.7 ± 0.1	82.11 ± 33.14	1.4 ± 0.6
Raven	90.85 ± 26.74	0.216 ± 0.002	93.68 ± 17.25	0.54 ± 0.01	95.92 ± 13.96	1.09 ± 0.01
GNN	99.20 ± 1.86	0.5 ± 0.1	93.51 ± 18.31	1.4 ± 0.3	95.73 ± 13.90	3.0 ± 0.4

a dataset of assembly graphs based on reads sampled from human genomic data on which the developed model was trained and evaluated. The model was also evaluated against a naive greedy approach, an exhaustive search using the positional information of reads, and an existing genome assembler Raven [6]. It was shown that it consistently outperforms the greedy approach in terms of reconstructed length, and is usually on-par with Raven while having no significant computational overhead. More interestingly, it was shown that the model outperformed Raven when given a highly complex graph from a repetitive region.

These results are particularly promising to solve challenging regions more accurately than the existing assemblers can. Future work will investigate the proposed technique on sequenced instead of generated data, with the ultimate aim of using it as a tool for untangling the assembly graphs to reduce fragmentation. We believe that combining path-finding techniques with deep learning will play a major role in improving *de novo* genome assembly.

ACKNOWLEDGMENT

Lovro Vrček has been supported by "Young Researchers" Career Development Program DOK-2018-01-3373, ARAP scholarship awarded by A*STAR, and core funding of Genome Institute of Singapore, A*STAR. Xavier Bresson has been supported by NRF Fellowship NRFF2017-10 and NUS-R-252-000-B97-133. Martin Schmitz has been supported by SINGA scholarship awarded by A*STAR. Mile Šikić has been supported in part by the European Union through the European Regional Development Fund under the grant KK.01.1.1.01.0009 (DATACROSS), by the Croatian Science Foundation under the project Single genome and metagenome assembly (IP-2018-01-5886), and by the core funding of Genome Institute of Singapore, A*STAR.

REFERENCES

[1] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh *et al.*, "Initial sequencing and analysis of the human genome," 2001.

[2] A. M. Wenger, P. Peluso, W. J. Rowell, P.-C. Chang, R. J. Hall, G. T. Concepcion, J. Ebler, A. Fungtammasan, A. Kolesnikov, N. D. Olson *et al.*, "Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome," *Nature biotechnology*, vol. 37, no. 10, pp. 1155–1162, 2019.

[3] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes *et al.*, "Nanopore sequencing and assembly of a human genome with ultra-long reads," *Nature biotechnology*, vol. 36, no. 4, pp. 338–345, 2018.

[4] S. Nurk, S. Koren, A. Rhie, M. Rautiainen, A. V. Bzikadze, A. Mikheenko, M. R. Vollger, N. Altemose, L. Uralsky, A. Gershman, S. Aganezov, S. J. Hoyt, M. Diekhans, G. A. Logsdon, M. Alonge, S. E. Antonarakis, M. Borchers, G. G. Bouffard, S. Y. Brooks, G. V. Caldas, H. Cheng, C.-S. Chin, W. Chow, L. G. de Lima, P. C. Dishuck, R. Durbin, T. Dvorkina, I. T. Fiddes, G. Formenti, R. S. Fulton, A. Fungtammasan, E. Garrison, P. G. Grady, T. A. Graves-Lindsay, I. M. Hall, N. F. Hansen, G. A. Hartley, M. Haukness, K. Howe, M. W. Hunkapiller, C. Jain, M. Jain, E. D. Jarvis, P. Kerpedjiev, M. Kirsche, M. Kolmogorov, J. Korlach, M. Kremitzki, H. Li, V. V. Maduro, T. Marschall, A. M. McCartney, J. McDaniel, D. E. Miller, J. C. Mullikin, E. W. Myers, N. D. Olson, B. Paten, P. Peluso, P. A. Pevzner, D. Porubsky, T. Potapova, E. I. Rogaev, J. A. Rosenfeld, S. L. Salzberg, V. A. Schneider, F. J. Sedlazeck, K. Shafin, C. J. Shew, A. Shumate, Y. Sims, A. F. A. Smit, D. C. Soto, I. Sović, J. M. Storer, A. Streets, B. A. Sullivan, F. Thibaud-Nissen, J. Torrance, J. Wagner, B. P. Walenz, A. Wenger, J. M. D. Wood, C. Xiao, S. M. Yan, A. C. Young, S. Zarate, U. Surti, R. C. McCoy, M. Y. Dennis, I. A. Alexandrov, J. L. Gerton, R. J. O'Neill, W. Timp, J. M. Zook, M. C. Schatz, E. E. Eichler, K. H. Miga, and A. M. Phillippy, "The complete sequence of a human genome," *bioRxiv*, 2021. [Online]. Available: https://www.biorxiv.org/content/early/2021/05/27/2021.05.26.445798

[5] H. Li, "Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences," *Bioinformatics*, vol. 32, no. 14, pp. 2103–2110, 2016.

[6] R. Vaser and M. Šikić, "Time-and memory-efficient genome assembly with raven," *Nature Computational Science*, vol. 1, no. 5, pp. 332–336, 2021.

[7] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.

[8] T. Freeman, S. Horsewell, A. Patir, J. Harling-Lee, T. Regan, B. B. Shih, J. Prendergast, D. A. Hume, and T. Angus, "Graphia: A platform for the graph-based visualisation and analysis of complex data," *bioRxiv*, 2020.

[9] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann *et al.*, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688–702, 2020.

[10] P. Gainza, F. Sverrisson, F. Monti, E. Rodola, D. Boscaini, M. Bronstein, and B. Correia, "Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning," *Nature Methods*, vol. 17, no. 2, pp. 184–192, 2020.

[11] G. Gonzalez, S. Gong, I. Laponogov, M. Bronstein, and K. Veselkov, "Predicting anticancer hyperfoods with graph convolutional networks," *Human Genomics*, vol. 15, no. 1, pp. 1–12, 2021.

[12] L. Vrček, P. Veličković, and M. Šikić, "A step towards neural genome assembly," *arXiv preprint arXiv:2011.05013*, 2020.

[13] H. Cheng, G. T. Concepcion, X. Feng, H. Zhang, and H. Li, "Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm," *Nature Methods*, vol. 18, no. 2, pp. 170–175, 2021.

[14] X. Bresson and T. Laurent, "A two-step graph convolutional decoder for molecule generation," *arXiv preprint arXiv:1906.03412*, 2019.

[15] C. K. Joshi, T. Laurent, and X. Bresson, "An efficient graph convolutional network technique for the travelling salesman problem," *arXiv preprint arXiv:1906.01227*, 2019.
[16] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.
[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.