A step towards neural genome assembly

Anonymous Author(s)
Affiliation
Address
email

Abstract

De novo genome assembly focuses on finding connections between a vast amount of short sequences in order to reconstruct the original genome. The central problem of genome assembly could be descried as finding a Hamiltonian path through a large directed graph with a constraint that an unknown number of nodes and edges should be avoided. However, due to local structures in the graph and biological features, the problem can be reduced to graph simplification, which includes removal of redundant information. Motivated by recent advancements in graph representation learning and neural execution of algorithms, in this work we train the MPNN model with max-aggregator to execute several algorithms for graph simplification. We show that the algorithms were learned successfully and can be scaled to graphs of sizes up to 20 times larger than the ones used in training. We also test on graphs obtained from real-world genomic data—that of a lambda phage and E. coli.

1 Introduction

Recently, we have witnessed a wide adoption of machine learning in graph representations and approaches to problem-solving on graph-structured data. In the center of this research are usually graph neural networks (GNNs) [1, 2] which are applied to various problems on graphs, such as finding the shortest path [3, 4], traveling salesman [5, 6], and bipartite matching [7]. Many of the earlier approaches, however, give the model complete freedom to determine how to obtain the solution from raw inputs, by using only the final algorithmic solutions as the supervision signal.

This was addressed by Veličković et al. [8], where the authors have provided explicit step-by-step guidance for learning algorithm execution. Moreover, they recognized that many of the classical algorithms share related subroutines, which enabled them not only to teach a model to execute several algorithms simultaneously, but also to demonstrate positive knowledge transfer between learning different algorithms.

While all the aforementioned research, and plenty of other work, made a significant breakthrough in the domain of machine learning on graph-structured problems, not a lot of work has yet been done on real-world systems. In this work, motivated by the contribution of Veličković et al. [8], and application to bipartite matching by Georgiev and Lió [7], we take a step in that direction and apply the neural graph algorithm execution approach on the problem of *de novo* genome assembly, which to this day remains as one of the most difficult problems in bioinformatics. *De novo* genome assembly aims at reconstructing the original genome sequence from hundreds of thousands of relatively short sequences called reads, without any knowledge of what the original sequence looks like. Ability to reconstruct genomes with high accuracy would have numerous applications in biology and precision medicine. In the ideal case, the problem is based on finding a Hamiltonian path through a complex directed graph, which is known to be an NP-complete problem. Due to long repetitive regions in genomes and artifacts that arise during sample preparation or sequencing, some nodes and edges should not be included in the final path. However, as a result of numerous reads covering the same

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

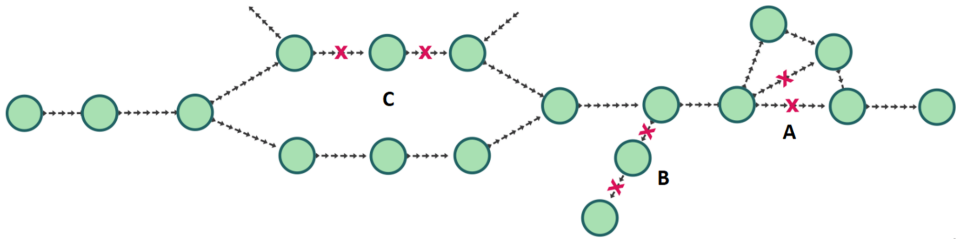

Figure 1: Example of structures in the assembly graph, before all the simplification steps. Letter **A** marks transitive edges, a short tip is marked with **B**, and a bubble which cannot be fully resolved is marked with **C**. Red crosses show which edges can be removed from the assembly graph.

region in the genome and biological knowledge, graphs can be simplified in several steps by removing redundant nodes and edges from them.

Here, we construct a model which can learn three of these simplification algorithms simultaneously and perform them in parallel, thus paving the way towards a purely neural execution of genome assembly.[1]

2 Problem setup

2.1 Genome assembly

After the reads are obtained from a sequencer, one of the most common approaches for *de novo* genome assembly nowadays is called Overlap-Layout-Consensus (OLC) paradigm.

The overlap phase consists of mutually overlapping all the reads in the sample. Reads which are contained in the others are discarded immediately. From the rest of them an assembly graph is built, where nodes depict reads and edges depict the overlaps between them. Following this is the layout phase, where a path approximating the genome has to be found. However, due to abundance of reads some of them are redundant, and due to sequencing errors some overlaps are false. Thus, structures in the graph corresponding to such occurrences have to be detected and removed. OLC-based state-of-the-art assemblers such as Raven [9], focus on three such structures—transitive edges, tips, and bubbles, which are the most common structures in the assembly graphs and can be seen in Figure 1. It is important to note that parts of the assembly graphs are paths even before the simplification, which makes the problem easier. Nevertheless, there exist regions of high complexity, which cannot be solved with the simplification algorithms described here.

After a few other processing steps, a path representing the assembly sequence is found. In the final phase, consensus, the assembly sequence is compared to the reads in the sample and polishing of the sequence is performed.

The transitive reduction is the first simplification performed. Given a triplet of nodes (n_i, n_j, n_k), edge $n_i \rightarrow n_k$ is transitive if there exist edges $n_i \rightarrow n_j$ and $n_j \rightarrow n_k$. Note that a transitive edge of one triplet can serve as a non-transitive in another triplet, and that the edge that "jumps over" more than one node is not transitive by this definition.

The second step is the removal of tips or dead ends—branches of several nodes that stem from the main chain and then cease abruptly, either due to sequencing errors or low sequencing coverage. They can be detected by running a depth-first search from the nodes with no outgoing edges and stopping once we find a node with multiple incoming or outgoing edges.

Finally, bubbles are the most complex structures that have to be removed. They are an arrangement of multiple paths connecting two nodes, where only one of these paths can be retained. However, nodes in those paths can be connected to other parts of the graph and removing them could result in a less accurate and more fragmented assembly sequence. This can be seen in Figure 1 in the top path of the bubble, where not all the edges can be removed.

[1] Code: `https://anonymous.4open.science/r/445a3c32-1dae-42e7-8079-d300a71b5a26/`

2.2 Neural execution of simplification algorithms

We model all three of the algorithms as graph traversal problems, where each of them has specific constraints on which edges can be traversed. The GNN model we use to learn these algorithms takes $T \in \mathbb{N}$ graphs $G = (V, E)$, where each node $i \in V$ in the graph has associated node features $\vec{x}_i^{(t)} \in \mathbb{R}^{N_x}$ and each edge (i, j) has associated edge features $\vec{e}_{ij}^{(t)} \in \mathbb{R}^{N_e}$. Here, N_x is the dimension of the node features, N_e is the dimension of the edge features, and $t \in \{1, \dots T\}$ is a timestep in the execution as well as the index of the graph that is being processed. For each graph, a node-level predictions are made, simulating the next step in the algorithm execution.

The structure of the model follows the *encode-process-decode* paradigm [10], where encoder and decoder networks are specific for each algorithm, and the processor network is algorithm-agnostic. This enables us to learn multiple algorithms simultaneously. We also include the termination network, which predicts whether each algorithm should continue with the execution, as is done in [8] where more information about the whole process can be found.

In short, this pipeline can be described with the following equations:

$$\vec{z}_i^{(t)} = f_A\left(\vec{x}_i^{(t)}, \vec{h}_i^{(t-1)}\right), \tag{1}$$

$$\mathbf{H}^{(t)} = P\left(\mathbf{Z}^{(t)}, \mathbf{E}^{(t)}\right), \tag{2}$$

$$\vec{y}_i^{(t)} = g_A\left(\vec{z}_i^{(t)}, \vec{h}_i^{(t)}\right), \tag{3}$$

$$\tau^{(t)} = \sigma\left(T_A\left(\overline{\mathbf{H}^{(t)}}\right)\right). \tag{4}$$

Here, f_A is the encoder network, P is the processor network, g_A is the decoder network, while the T_A is the termination network. Latent features for each node are given with $\vec{h}_i^{(t)}$, and the input to the termination network is the mean latent state across all the nodes—effectively giving the latent state of the whole graph.

Since we want to simulate algorithms as accurately as possible the supervision signal is given at each step of execution, thus training the model by teacher forcing.

3 Results and discussion

First, we have learned our model to execute all three of the mentioned algorithms separately. For training the model on each of them, 100 graphs had been generated. Each graph consisted of a path graph that is 50 nodes long, with additional structures we want the network to recognize added to that path graph. Since algorithms have strict constraints on characteristics of these structures, e.g. a branch is a dead end only of it consists of five nodes or less, we generated not only the structures that have to be recognized as problematic regions but also those which should not be recognized as such—branches longer than five nodes and transitive edges that connect nodes that are more than one node apart. The positions of such structures and their sizes have been randomly generated for each graph. Testing was performed on graphs that were the same size, but also 2, 4, 8, and 20 times larger, which can be seen in Table 1.

All the algorithm-dependent networks in our model are simple linear projections. For processor network we use an MPNN with a max-aggregator. This architecture which was shown to have better performance than MPNNs with other types of aggregators and graph attention networks (GAT) [11]. Message function used in the MPNN is also linear projections and update function is linear projection followed by a gated recurrent unit (GRU) [12]. We use Adam optimizer [13] with a learning rate of 1e-5, and early stopping with the patience of 10 epochs. The latent dimension of $K = 32$ was used both for latent features and encoded node and edge features.

After proving the network can learn all three of the simplification algorithms, we train the network to execute them simultaneously. The obtained accuracy is similar to the case of isolated learning for all three algorithms, from which we conclude that the network has successfully learned to execute them in parallel. Results can be seen in Table 2.

Finally, we test the parallel execution model on real-world data—we use lambda phage (*Escherichia virus Lambda*) and *Eschericia coli* genomic data sequenced with Oxford Nanopore Technologies

Table 1: Scaling of algorithm execution for isolated learning of algorithms.

	Scaling				
Algorithm	1x	2x	4x	8x	20x
Transitive removal	98.10%	99.00%	99.52%	99.76%	99.91%
Tips trimming	98.05%	98.96%	99.49%	99.70%	99.87%
Bubble popping	98.16%	99.03%	99.53%	99.77%	99.90%

Table 2: Scaling of algorithm execution for parallel learning of algorithms.

	Scaling				
Algorithm	1x	2x	4x	8x	20x
Transitive removal	98.21%	99.07%	99.50%	99.89%	99.92%
Tips trimming	98.45%	99.11%	99.46%	99.76%	99.89%
Bubble popping	98.17%	99.02%	99.51%	99.78%	99.90%

sequencer. The assembly graph was produced with Raven assembler [9]. We choose lambda phage because the graphs produced are small and convenient for more in-depth analysis, whereas E. coli is larger and gives better insight into how our model would generalize to eukaryotic genomes. Specifically, lambda phage graph consists of only 60 nodes, whereas E. coli graph consists of around 3000 nodes. The results can be seen in Table 3.

Table 3: Parallel algorithm execution on the assembly graph of lambda phage.

	Transitive removal	Tips trimming	Bubble popping
Lambda phage	98.04%	93.33%	97.47%
E. coli	99.67%	98.84%	99.26%

The results on lambda phage are somewhat lower than on the generated graphs, especially for the trimming of dead ends. This can mostly be contributed to the fact that structures found in real-world assembly graphs are often significantly more complex than the structures found in the graphs generated for the training purposes. While the network was able to learn how to deal with transitive edges, dead ends, and bubbles, it was not exposed to structures which cannot be solved by the algorithms described here. Since the lambda phage graph is small, a single such structure can be detrimental to achieving high accuracy. On the other hand, E. coli graphs are large compared to the others used in this work. We have already shown that the model achieves higher accuracy on larger graphs which is due to parts of graphs being paths without junctions. The same is true for E. coli graph, which makes some predictions easy; thus, the accuracy is considerably higher than in the case of lambda phage.

Our next step is to generalize model to solve complex structures which often cannot be solved exactly. Most of the state-of-the-art methods solve this part by manually set parameters of heuristic algorithms usually overfitting to several known model organisms, such as human. A neural model trained on complex and diverse dataset could simplify the genome assembly for less-known species. This urges us to train models on such dataset, which we leave to future work.

4 Conclusion

In this work we have successfully trained a neural model to execute three algorithms used for the graph simplification in *de novo* genome assembly, which remains one of the most difficult algorithmic problems to this day. Although the learned algorithms are simple, they are an important part of the graph simplification phase in the assembly process. Furthermore, we prove this method is suitable for this type algorithms and problems. After showing that the model is able to perform on graphs multiple times larger than the ones in the training set, we also show that it produces accurate predictions on real-world data. We believe this is a step towards automating the process of genome assembly, which will reduce the need for heuristics and hand-picked parameters.

References

[1] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[2] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

[3] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[4] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

[5] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700, 2015.

[6] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[7] Dobrik Georgiev and Pietro Lió. Neural bipartite matching. *arXiv preprint arXiv:2005.11304*, 2020.

[8] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. *arXiv preprint arXiv:1910.10593*, 2019.

[9] Robert Vaser and Mile Sikic. Raven: a de novo genome assembler for long reads. *bioRxiv*, 2020.

[10] Jessica B Hamrick, Kelsey R Allen, Victor Bapst, Tina Zhu, Kevin R McKee, Joshua B Tenenbaum, and Peter W Battaglia. Relational inductive bias for physical construction in humans and machines. *arXiv preprint arXiv:1806.01203*, 2018.

[11] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[12] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.