Parallelized algorithms for identifying and analyzing significant multi-step pathways in fly connectome data

Jack Lindsey
jwl2182@cumc.columbia.edu

1 Background

Recent advances in connectomics have enabled the construction of a nearly complete map of neuronal connectivity in the fly central brain, containing roughly $N = 25,000$ neurons and 20 million synapses (connections between neurons) (Scheffer et al., 2020). The unprecedented scale and detail of this data presents a computational challenge: how can we make sense of this neural circuitry and understand its function? Two features of the connectome data make this a challenging problem. First, connectivity is very dense: a given neuron often receives inputs from, and projects outputs to, thousands of other neurons. Second, many pairs of neurons interact strongly via indirect (multi-synapse) pathways. Characterizing the interactions between neurons in principle requires consideration of $O(N^k)$ paths, where N is the number of synapses and k is the maximum path length under consideration. In this project we develop and applying algorithms for addressing this problem, focusing on methods that are highly parallelizable and scalable.

In our lab, we are particularly interested in the mushroom body, a key site of learning and memory in the fly brain. Dopaminergic neurons (DANs) in the mushroom body, which modulate synaptic plasticity, receive highly diverse inputs from across the brain (Li et al., 2020). Untangling the pathways that drive DAN activity will allow us to make predictions about how flies learn sensory associations. In this project we apply our algorithms to analyzing pathways from mushroom body output neurons (MBONs) to DANs, which comprise a surprisingly large fraction of DAN input. These pathways allow learning to be driven recursively by learned responses and associations, which enables higher-order behaviors like second-order conditioning and memory consolidation.

2 Relevant prior work

2.1 In connectomics

Methods for analyzing fly connectome data are in fairly early stages. Typically, the number ("count") of synapses between neurons is used as a proxy for their effective connection strength. Often, analyses consider normalized connectivity, such that the inputs to a given neuron sum to 1. Based on this assumption (Li et al., 2020) introduced the notion of effective strength of multi-step pathways, computed by multiplying the normalized strength of each connection along the pathway. That is, if neuron A receives 10% of its input synapses from B, and B receives 10% of its inputs from C, A can be considered to receive 1% of its effective two-step input from C via B. Li et al. (2020) use this approach to quantify multi-step connectivity strength between pairs of neurons, and to identify the most important intermediate neurons (those that "carry" the greatest amount of connectivity strength) in two-step pathways. Schlegel et al. (2021) use a sampling-based approach to tracing "information flow" through the connectome, starting from a given source set of sensory neurons. Their strategy is to begin with a pool consisting of the source neurons, and sample neurons with probability according to the fraction of inputs they receive from the current pool. The number of iterations that pass before a given neuron is added to a pool can be thought of as its "layer" in the flow of information from the source set.

2.2 In graph algorithms

Many algorithms exist to find important paths in (weighted, directed) graphs. A classic family of such algorithms are shortest path algorithms, like Dijkstra's algorithm, versions of which can run in $O(E + V \log V)$ time where E andn V are the number of edges and vertices in the graph, respectively (Fredman and Tarjan, 1987). A standard extension to the shortest path problem is that of finding the top k shortest paths between a source and target vertex, such as Yen's algorithm (Yen, 1970) and extensions (Eppstein, 1998; Hershberger et al., 2007), including parallel implementations (Ruppert, 2000), which typically introduce a factor of k into the time complexity.

The problem of finding significant paths in connectome data can be reconstrued as a shortest paths problem by defining edge weights between nodes i and j as proportional to $(-\log w^{ij}_{norm})$ where w^{ij}_{norm} is the normalized synaptic count between neurons i and j. In this formulation the length of a path varies monotonically with the effective strength according the definition above.

Related to the problem of finding significant paths between nodes is that of identifying important intermediate nodes along such paths. The property of *betweenness centrality* has been introduced to quantify this notion. The betweenness centrality of a given node v is typically defined as the fraction of shortest paths (between the source(s) and target(s) of interest) that pass through v (Freeman, 1977).

2.3 Shortcomings of existing approaches

In the connectome data, inputs to neurons are highly distributed. Consequently, a single path typically makes up a small fraction of the total effective interaction between a given source and target neuron. Even the top k such paths will constitute a small total contribution, for manageable values of k. We would like a centrality metric that more gracefully accounts for the cumulative contributions of many weak paths that all pass through a given neuron when computing that neuron's centrality.

Moreover, we may be interested in a comprehensive summary of the pathways by which a given source neuron affects a given target neuron. Algorithms based on computing strongest paths will only return a few such paths. However, enumerating all paths up to length k and computing their effective strengths requires listing $O(V^k)$ paths and performing $O(k)$ multiplications for each, which is intractable for $V \sim 25,000$ for values of k larger than about 2.

3 Approach

3.1 Sampling-based

Algorithm 1 Path sampling algorithm

```
Initialize $PATHS = []$
Initialize $CANDIDATES$ with $M$ uniform samples from $SOURCES$
Initialize k = 1
while $k \leq k_{MAX}$ do
    for $p \in CANDIDATES$ do
        Remove $p$ from CANDIDATES
        for $n \in NODES$ do
            if $rand() < w_{norm}^{n,p[-1]}$ then
                Add $p' = Concat(p, [n])$ to CANDIDATES
                if $n \in SOURCES$ then
                    Add $p'$ to $PATHS$
                end if
            end if
        end for
    end for
end while
Return $PATHS$
```

We would like to design an algorithm with the property that paths are sampled proportionally to

$$p([n_1, n_2, ..., n_k]) \sim \prod_{i=1}^{k-1} w_{norm}^{n_i,n_{i+1}}$$

A sampling-based approach sacrifices exactness (by introducing stochasticity into the sampled path distribution) in favor of efficiency, scalability, and the potential for parallelization, as samples can be collected in parallel. We can trade off computation time for accuracy by collecting more or fewer samples.

Our approach is as follows, described in detail in Algorithm 1. We maintain a set of sampled paths and construct them in stages. We begin with a pool of size M sampled uniformly from the set of target nodes of interest (in our case, the 322 DANs in the connectome data) and probabilistically extend each path to include neurons presynaptic to the most recent in the path, with probability proportional to the normalized connection strength between those nodes. When a neuron from the source population of interest (in our case, the 70 MBONs in the connectome data), the path is no longer extended and is returned. If a path reaches a certain maximum length (in our case, length 10) without containing a source node, it is discarded and not returned. In expectation, the number of returned paths will be M due to the input normalization.

Since the construction of each path is independent of other path candidates with a different initial node (or set of nodes), this algorithm can be readily parallelized by dividing up the initial $CANDIDATES$ pool across processors.

3.2 Vectorized centrality computation

For some applications, we may not be interested in the details of the pathways connecting a set of source and target neurons, and can settle for a summary statistic describing the centrality of intermediate nodes. Here we define the centrality of a neuron n along paths of length k between a given set of source and target neurons as follows:

Algorithm 2 Vectorized centrality algorithm

```
**Initialize** normalized adjacency matrix $W$
**Compute** $W^k$ for $k \in \{1, 2, ..., k_{max}\}$
**for** $k \in \{1, 2, ..., k_{max}\}$ **do**
    **Compute** $(c_n^k)_{in} = \sum_{s \in sources} (W^k)_{s,n}$ for all $n$
    **Compute** $(c_n^k)_{out} = \sum_{t \in targets} (W^k)_{n,t}$ for all $n$
**end for**
**for** $k \in \{2, ..., k_{max}\}$ **do**
    **Initialize** $c_n^k = 0$ for all $n$
    **for** $k_{in} \in \{1, 2, ..., k_{max} - 1\}$ **do**
        $k_{out} = k_{max} - k_{in}$
        **Compute** $c_n^k = c_n^k + \frac{1}{k}(c_n^{k_{in}} \cdot c_n^{k_{out}})$ for all $n$
    **end for**
**end for**
**Return** $c_n^k$ for all $n$
```

$$c_n^k = \frac{1}{k} \sum_{paths=[n_1,...,n_k] \text{ s.t. } n \in path} \left(\prod_{i=1}^{k-1} w_{n_i,n_{i+1}} \right)$$

where the normalization by $\frac{1}{k}$ is meant to reflect that each node in a k-length path bears only a fraction of responsibility for the information propagation along that path.

To perform this computation, we precompute

$$(c_n^{k'})_{in} = \sum_{paths=[n_1,...,n_{k'}=n]} \left(\prod_{i=1}^{k-1} w_{n_i,n_{i+1}} \right)$$

And

$$(c_n^{k'})_{out} = \sum_{paths=[n_1=n,...,n_{k'}]} \left(\prod_{i=1}^{k-1} w_{n_i,n_{i+1}} \right)$$

Then we note that

$$c_n^k = \sum_{k_1+k_2=k} c_{in}^{k_1} \cdot c_{out}^{k_2}$$

The detailed algorithm is described in Algorithm 2. This algorithm is highly amenable to parallelization, as the computation of the $(c_n^{k'})_{in}$ and $(c_n^{k'})_{out}$ variables may be construed as a matrix multiplication, and the ultimate computation of c_n^k as an element-wise dot product.

4 Results and verification

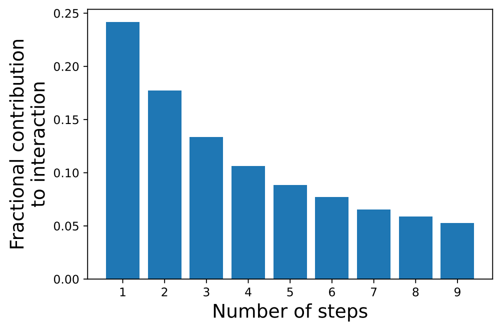

Figure 1: Contribution of each path length to the interaction between MBONs and DANs, as computed by the sampling algorithm.

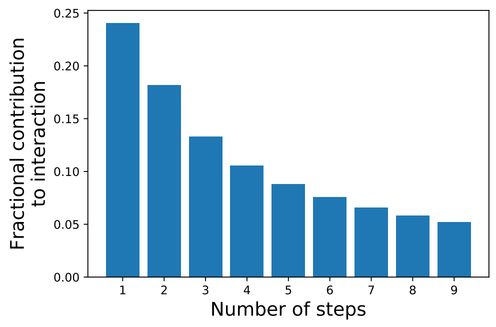

Figure 2: Contribution of each path length to the interaction between MBONs and DANs, as computed by the vectorized centrality algorithm.

Many analyses can be conducted with the results of the algorithms described above, but we focus on two example use cases here. One summary statistic of interest is the extend to which different path lengths mediate the interaction between the set of source and target nodes. Do MBONs communicate with DANs primarily via direct synaptic connections, or primarily by two-step pathways, three-step pathways, etc.? With a pool of sampled paths we can address this question simply by counting the fraction of paths of each length. Results are shown in Figure 1.

We can also address the same question using intermediate results from the centrality algorithm. The contribution of each path length can be computed as

$$Contribution(k) = \sum_{s \in sources} \sum_{t \in targets} (W^k)_{s,t}$$

The results of this computation are shown in Figure 2. The consistency between Figures 1 and 2 serves as verification of correct implementation of both algorithms.

Second, we look at the contribution of different neuron types to the interactions. Specifically, we compute the quantity $\frac{1}{k_max} \sum_{k=1}^{k_{max}} c_n^k$ for each neuron n and group the results by the known cell type of each neuron. The results of the vectorized centrality algorithm give us this capability directly. For the sampling algorithm, we simply count the fraction of paths of each length for which a given neuron n is present. The results are shown in Figure 3 (for the sampling algorithm) and Figure 4 (for the vectorized algorithm), respectively. Once again, the results are highly consistent, verifying correct implementation.

5 Scaling analyses

Although the fly connectome data is of fixed size, future datasets (e.g. of other species) may be orders of magnitude larger, and hence scalability is important to addressing such datasets. We tested the scaling properties of the sampling algorithm by varying the size of the graph on which the path sampling was conducted. To do so we simply subsampled random subsets of neurons and considered the subgraph induced by them when running the algorithm. We also analyzed the role of parallelization in the algorithm's runtime on an 8-core Macbook Pro laptop by varying the number of processes used to parallelize the algorithm. The results are shown in Figures 5 and 6. For large enough graph sizes, runtime appears to scale slightly superlinearly

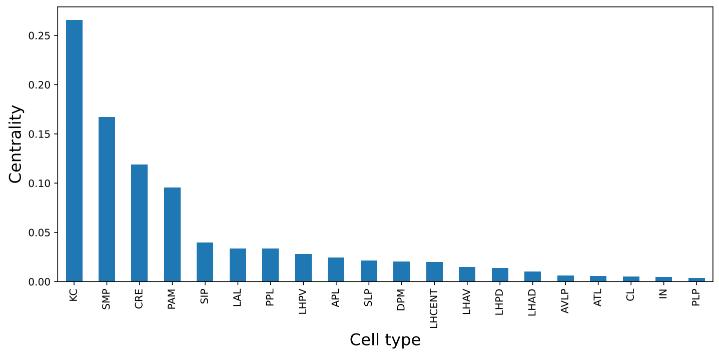

Figure 3: Average (across path lengths) centrality of different cell types in multi-step MBON-DAN interactions, using the sampling algorithm.

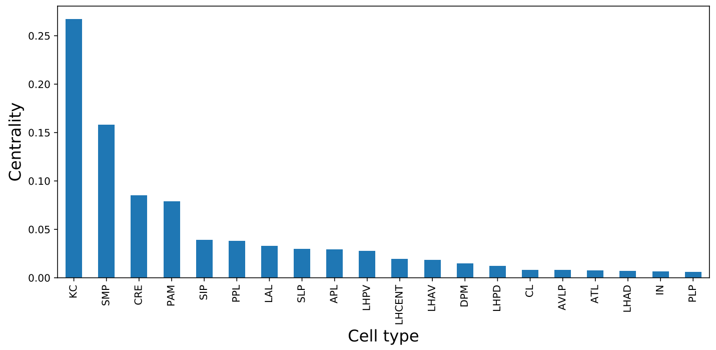

Figure 4: Average (across path lengths) centrality of different cell types in multi-step MBON-DAN interactions, using the vectorized centrality algorithm.

with the number of nodes. The run time is cut down approximately linearly as the number of processes is increased, up until a saturation point around 4 processes. We presume that gains could be achieved at higher process counts on a machine with more physical cores.

Next, we analyze the scaling of the matrix-multiplication-based centrality algorithm. We used three separate implementations. The first was a serial CPU implementation using numpy's built-in matrix operations. The latter two made use of PyTorch, a machine learning library which, among other things, facilitates parallelization of matrix operations (Paszke et al., 2017).

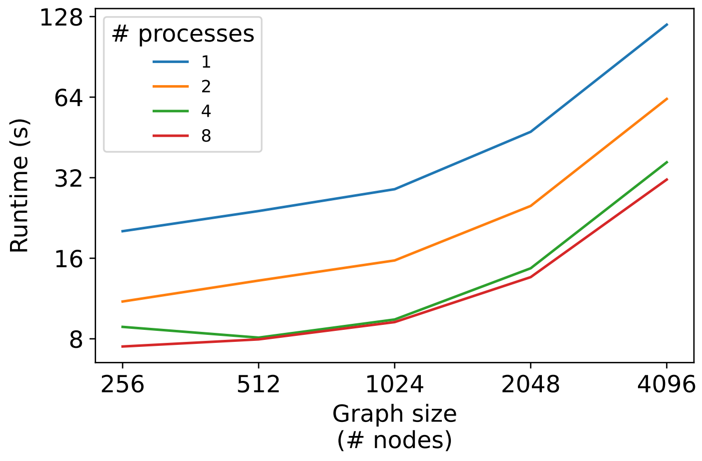

Figure 5: Flower one.

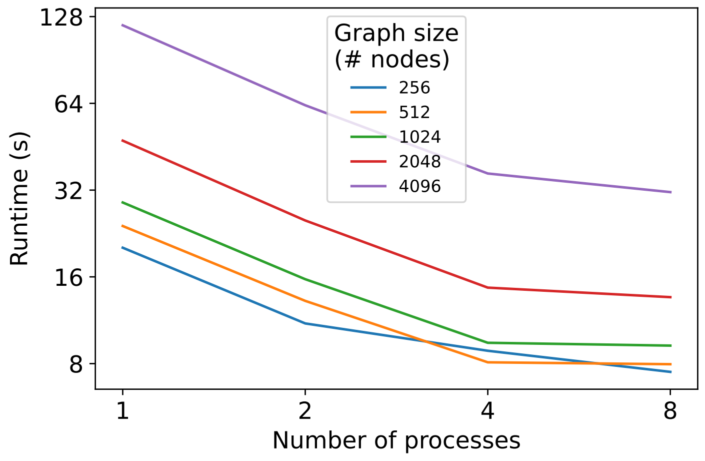

Figure 6: Flower two.

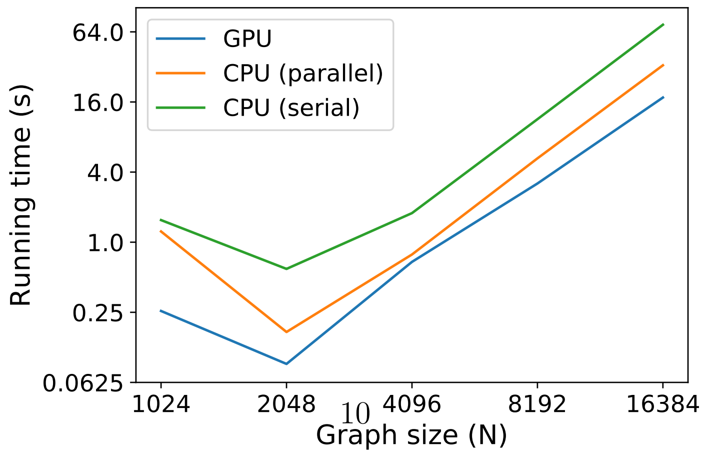

Figure 7: Flower two.

We experimented with a PyTorch-based parallelized CPU implementation (on 8 cores) and GPU implementation (NVIDIA 2080 ti, CUDA 10.1). We found (see Figure 7) a scaling of runtime with graph size that was between quadratic and cubic. We also found a notable performance improvement from CPU parallelization, and an even larger improvement from GPU utilization.

6 Conclusion

In conclusion, we introduced two new algorithms for analyzing multi-step pathways mediating interactions between a set of source and target nodes of interest in connectomic graph data. We developed parallelized implementations of these algorithms and applied them to compute certain summary statistics of the profile of interactions between mushroom body output neurons and dopamine neurons in the fly connectome. We verified that our two algorithms gave converging results where applicable, and conducted scaling analyses demonstrating the benefits of parallelization and the potential for even further benefits on more advanced hardware.

7 Code details

All code required to produce the figures in this report is provided in Jupyter notebooks, included in the submission, which can be run using Python version 3.7.3. The core of the sampling algorithm is implemented in sampling_utils.py, and the code used to apply the algorithms is presented in DAN_MBON_paths_analysis.py (for general data loading and the sampling algorithm analyses) and Matrix_alg.py (for code specific to analyses of the vectorized centrality algorithm). Running the code requires the navis (version 1.1.0) and neuprint-python (0.4.16) packages, which can be downloaded via pip. Running Matrix_alg.py requires Pytorch (version 1.6.0) compiled with CUDA (10.1) support, which is also available via pip.

References

Eppstein, D. (1998). Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673.

Fredman, M. L. and Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615.

Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41.

Hershberger, J., Maxel, M., and Suri, S. (2007). Finding the k shortest simple paths: A new algorithm and its implementation. *ACM Transactions on Algorithms (TALG)*, 3(4):45–es.

Li, F., Lindsey, J. W., Marin, E. C., Otto, N., Dreher, M., Dempsey, G., Stark, I., Bates, A. S., Pleijzier, M. W., Schlegel, P., et al. (2020). The connectome of the adult drosophila mushroom body provides insights into function. *Elife*, 9:e62576.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

Ruppert, E. (2000). Finding the k shortest paths in parallel. *Algorithmica*, 28(2):242–254.

Scheffer, L. K., Xu, C. S., Januszewski, M., Lu, Z., Takemura, S.-y., Hayworth, K. J., Huang, G. B., Shinomiya, K., Maitlin-Shepard, J., Berg, S., et al. (2020). A connectome and analysis of the adult drosophila central brain. *Elife*, 9:e57443.

Schlegel, P., Bates, A. S., Stürner, T., Jagannathan, S. R., Drummond, N., Hsu, J., Capdevila, L. S., Javier, A., Marin, E. C., Barth-Maron, A., et al. (2021). Information flow, cell types and stereotypy in a full olfactory connectome. *Elife*, 10:e66018.

Schrijver, A. (2003). *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media.

Yen, J. Y. (1970). An algorithm for finding shortest routes from all source nodes to a given destination in general networks. *Quarterly of Applied Mathematics*, 27(4):526–530.