Johns Hopkins University
Applied Physics Laboratory
APL BRAIN

Connectome subgraph isomorphisms and graph queries

Jordan K. Matelsky, Elizabeth P. Reilly, Erik C. Johnson, Brock A. Wester, William Gray-Roncal
Johns Hopkins University Applied Physics Laboratory; Laurel, Maryland
Correspondence: jordan.matelsky@jhuapl.edu, william.gray.roncal@jhuapl.edu

December 17, 2020

Abstract

As connectomics datasets continue to grow in size and complexity, methods to search for and identify interesting graph patterns offer a promising approach to quickly reduce data dimensionality and enable discovery. Recent advances in neuroscience have enabled brain structure exploration at the level of individual synaptic connections. These graphs are often too large to be analyzed manually, presenting significant barriers to searching for structure and testing hypotheses. We combine graph database and analysis libraries with an easy-to-use neuroscience grammar suitable for rapidly constructing queries and searching for subgraphs and patterns of interest. This abstracts many of the computer science and graph theory challenges associated with nanoscale brain network analysis and allows scientists to quickly achieve reproducible findings at scale. We demonstrate these tools to search for motifs on simulated data and real public connectomics datasets, and share simple and complex structures relevant to the neuroscience community. We contextualize these results and provide case studies to motivate future neuroscience questions. All of our tools are released open source to empower other scientists to use and extend these methods.

1 Introduction

Modern nanoscale-connectomics research commonly involves the conversion of microscopy imagery data into a graph representation of connectivity, where nodes represent neurons, and directed edges represent the synapses between them. This process enables researchers to convert terabytes or even petabytes of imagery into megabytes or gigabytes of graph data. This process reduces the cost and complexity of interrogating the data, at the expense of losing morphology information [1, 2]. Though this graph representation uses substantially less storage-space on disk, extracting knowledge about even the seemingly simplest network questions (e.g., identifying local graph structure around a particular neuron, or comparing the downstream targets of a certain cell type) may still exceed the computational power, timelines, and budgets available to many research teams, due to the exponential nature of many graph algorithms [3].

Connectomics researchers have begun to address this challenge by adopting existing large-scale graph management software, such as graph databases, and by enforcing consistent, well-architected data schemas [4, 5]. These systems provide performant and cost-effective ways to manipulate larger-than-memory graphs, but tend to require familiarity with complex and nuanced graph query programming languages such as Gremlin or Cypher. Though graph databases continue to grow in popularity, the expertise to administer or use these technologies is still not common in the biological sciences.

In order to make the study of connectomes accessible and computationally efficient, we developed *DotMotif*, an intuitive but powerful graph query language designed to reduce the expertise and time required to begin interrogating biological graphs of any size. DotMotif acts as an interface to common graph management systems such as the NetworkX Python library or the Neo4j graph database, abstracting the intricacies of subgraph-query design and enabling researchers to focus on field-specific scientific inquiry.

DotMotif assists researchers in query design by exposing an intuitive, simple syntax that automatically validates and optimizes user queries, and adapts to different graph tool backends without the need for

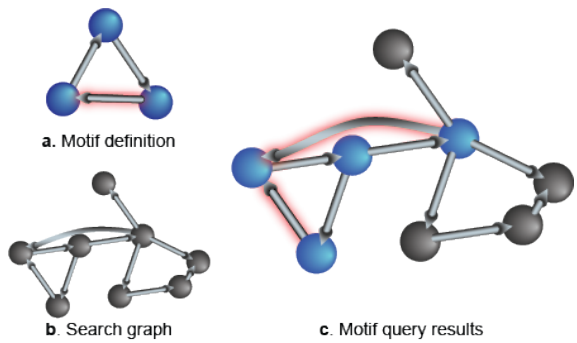

Figure 1: *Subgraph search.* Subgraph search is formally defined in the *Background* section. **a.** A motif query is defined. In this example, the query is a simple directed triangle. One edge has been assigned an attribute constraint (perhaps a neurotransmitter type or weight threshold). **b.** A directed search graph, or "host" graph. **c.** The motif is discovered in the search graph. Two unique detections are shown here, in which two of the node participants happen to be "reused".

additional user input.

We present the DotMotif Python package and software architecture, which enables researchers to write queries while remaining agnostic to underlying technologies. We demonstrate DotMotif's utility by comparing its advantages over manual query design, and we share several use-cases inspired by ongoing research in the connectomics community.

2 Background

Many of the fundamental questions of modern neuroscience rely upon the study of simple circuits of neural connectivity in the brain. These simple circuits might be likened to individual logic gates in a computer, in that they are hypothesized to be repeated and reused with conserved local network topology. Though they might comprise only a small number of neurons, as shown in **Figure 1**, these graph *motifs* may be critical to unraveling or discovering the existence of larger structures of the brain, such as cortical columns or modules of computational importance [6, 7, 8].

Identifying these motifs is one motivation for the field of *connectomics*, the study of the brain through the lens of its connectivity. Many related research efforts seek to construct graph representations of the brain, commonly represented by the notation $G = (V, E, A)$, where a graph G is made up of nodes V, representing neurons; (optionally directed) edges E, representing synapses between them; and arbitrary attributes A, which may be associated with vertices, edges, or the graph as a whole. For example, node attributes might include qualities such as cell type or functional information. Edge attributes might include synapse weight or neurotransmitter type. Graph attributes might include the species or individual from which the connectome was generated. Neuroscience questions may be reformulated as analyses on a graph, and the neuroscientist may add graph theory to the toolbox of strategies with which to understand the brain [6, 9]. Such questions include searching for specific subgraph structures, investigating the connectivity of specific neuron cell types or categories, proofreading connectomes for accuracy, and generating summary statistics on the graph as a whole [10, 11, 2, 12, 13, 14, 8].

Even some of the simplest graphs generated by the connectomics community in recent years have spanned multiple gigabytes of hard drive space [1, 8], rendering conventional graph toolkits, such as the common networkx library [15] (or its counterparts in other programming languages) under-powered to address the needs of the scientific community. These tools, which often require all graph data to be stored in RAM, would require impractically expensive compute hardware in order to run fully in-memory, and would require impractically long timelines in order to run while swapping data from memory to disk. Instead, some teams [4] have opted to leverage "out-of-memory" tools, such as Neo4j [5] or Cayley [16], graph databases that operate on graph data much like conventional relational databases operate on tabular data. Despite their power, such tools require expertise in specialized query languages such as Cypher or Gremlin, much like relational databases require knowledge of languages such as SQL. Developing or hiring for this sort of domain expertise may be impractical for many neuroscience research laboratories and is independent from from the core expertise needed to formulate and test neuroscience hypotheses. The need for computationally efficient, accessible, and intuitive neural graph analysis tools motivates this work.

3 Results

We share illustrative DotMotif experiments run on a set of publicly-available, seminal connectome datasets. We first validate our results by comparing against existing online data, and then share a network-analysis result on both partial as well as complete connectomes. Lastly, we share performance benchmarks for graphs of various sizes.

3.1 Datasets

We demonstrate the use of our tool on both near-complete (most or all of the organism's neurons are

included in the graph) connectomes as well as on partial connectomes. Here, we compare connectomes of the invertebrate *C. elegans* worm [17, 18], the partial connectome of the invertebrate fruit fly from the HHMI Janelia Hemibrain [1] project, and the partial vertebrate mouse visual cortex connectome from the Intelligence Advanced Research Project Activity (IARPA) Machine Intelligence from Cortical Networks (MICrONS) project [19, 20, 8]. These published graphs do not adhere to a particular database schema or storage system, and each has different node and edge attributes. Despite these differences, these graphs may all still be analyzed using the techniques we detail here.

3.2 Tool validation: Declarative queries

In order to illustrate compatibility with data stored in the neuPrint data format [4], we ran a DotMotif search query on the partial *Drosophila melanogaster* connectome (dubbed *"hemibrain"*) recently published by HHMI Janelia [1]. We then validated these results with the neuPrint API at *neuprint.janelia.org* [4].

In this example query, we wanted to find all *Antennal Lobe* inputs to *Kenyon Cells* with weights within a certain range:

DotMotif Syntax:

```
# Find synapses from ant. lobe to KCs:
src -> kenyon [weight>12, weight<=14]

# Set known cell-type attributes:
src.type contains "mALT"
kenyon.type contains "KC"
```

We identified `src`-neuron instance IDs of 1917188956, 1825085656, 5813063239, 1887163927, 1917188956, 1825085656, 1037293275, and were able to use DotMotif or other tools to interrogate additional cell metadata for these neurons.

In order to confirm that the query above returned identical results as the canonical neuPrint server, we validated these results with the following equivalent Cypher command in the neuPrint web application:

neuPrint Cypher Syntax:

```
MATCH
  (src:Neuron)-[
    synapseSet:ConnectsTo
  ]->(kenyon:Neuron)
WHERE src.type CONTAINS "mALT"
  AND kenyon.type CONTAINS "KC"
  AND synapseSet.weight <= 14
  AND synapseSet.weight > 12
RETURN DISTINCT src, kenyon
```

3.3 Undirected subgraph searches

As an illustration of the minimal configuration requirements of the DotMotif package, we searched the recently-published, human-proofread subgraph from the IARPA MICrONS project [19, 20, 8] for all undirected graphs of size $|V| \leq 6$, with ordering from the *Atlas of Graphs* [21, p. 8–30], and counted the number of times each subgraph appeared. For example, we counted all undirected triangles (*Atlas of Graphs* ID #7) and discovered that there were 6894 unique triangle monomorphisms in the MICrONS graph when ignoring edge direction. There were 123,264 unique undirected rectangle (*Atlas of Graphs* ID #16) monomorphisms. The only undirected graph in this set with *no* occurrences in the MICrONS graph was Motif ID #208, the complete graph of 6 nodes. We publish this complete dataset of all motifs and their respective counts for community use (see *Supplemental Materials*).

To contextualize these results, we calibrated the parameters of four random graph models (Erdős–Rényi [22], Undirected Geometric [23], Watts-Strogatz [24], and Barbarási-Albert [25]) to match the graph density (D=$\frac{2|E|}{|V|(|V|-1)}$) of each estimated connectome, and ran the same count of subgraph motifs on samples taken from each random graph distribution. We similarly performed the same subgraph motif counts on one more random graph model — a degree-preserving graph randomizer, as used in efforts such as [8] — which unlike the other models, begins with an existing connectome graph and perform arbitrary swaps of edges such that the overall in- and out-degrees of all nodes remains the same, but specific connectivity changes.[26]

The graph densities were computed as 0.03 for the MICRONS graph and 0.06 for the *C. elegans* graph. The code used to compute these values will be made available to the public (see *Supplemental Materials*). In order to compare these results with a complete connectome example dataset, we ran this count-scan on all chemical synapses of the hermaphrodite *C. elegans* connectome [17, 27, 18, 28] published by [18, 28] and available from *WormWiring.org*. All motifs tested occurred at least once in the *C. elegans* connectome. We discovered the top n most common motifs did not vary dramatically across biological connectomes and their random-graph counterparts, but the relative counts of each motif different dramatically based upon the connectome and motif in question (**Figure 2**).

Queries from this set of experiments were run using Neo4j, NetworkX, and GrandIso executors, as convenient, leveraging our ability to seamlessly switch between executors when using DotMotif.

All data and results from this study will be made publicly available as described in *Supplemental Mate-*

rials. These experiments may aid in the design or selection of random graph models to approximate a connectome graph, and in the design or selection of query graphs when studying reconstructed connectomes.

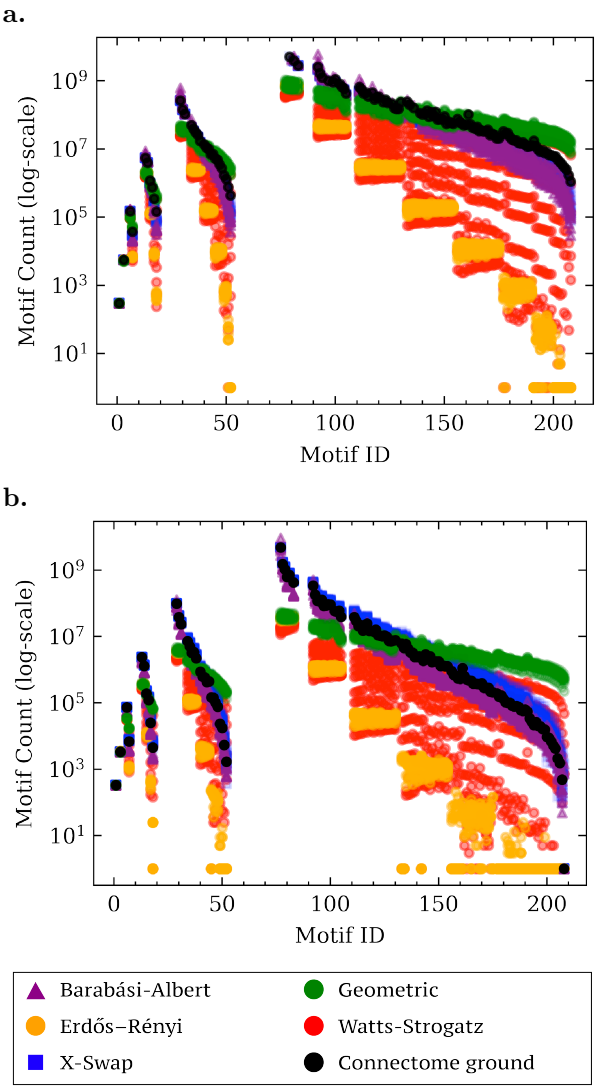

Figure 2: a & b: The count of every undirected subgraph of size $|V| \leq 6$. Each motif was counted in the connectomes, as well as in random graphs calibrated to match the density of each connectome. x-axis is the motif ID from the *Atlas of Graphs* text. Gaps indicate omitted motifs (those with multiple connected components). **a. Comparing *C. elegans* with random graphs calibrated to *C. elegans* density.** Due to its higher density, the *C. elegans* graph had higher motif counts than those of the MICrONS graph in general, despite a lower vertex-count. **b. Comparing MICrONS with random graphs calibrated to MICrONS density.** Full-resolution copies of this graphic are available online (see *Supplemental Materials*).

3.4 Directed subgraph searches

Recent work has investigated the relative prevalence of directed triangle motifs in brain graphs and the importance of such motifs for local computation [8, 29, 30]. Here we count all unique, directed, three-node motifs in the *MICrONS Phase 1* and Cook et al *C. elegans* connectomes.

As expected, lower-density motifs appear with greater frequency. Interestingly, the connectomes differed dramatically in both absolute count as well as in distribution shape, with many directed motifs occurring few to no times in the MICrONS graph (**Figure 3**).

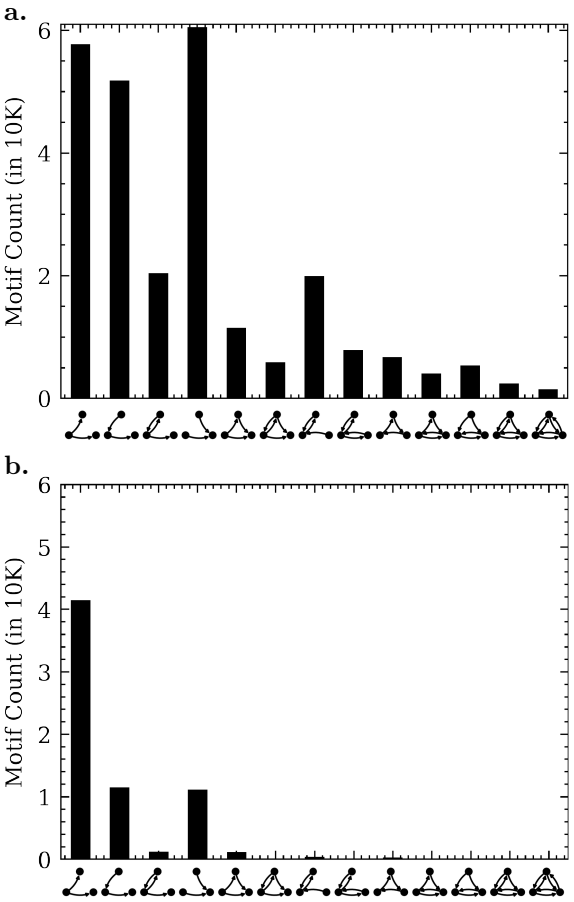

Figure 3: Quantities of directed triangles from the MICrONS Phase 1 graph and the Cook et al 2019 *C. elegans* chemical synapse connectome. **a. *Cook 2019 C. elegans* directed triangle counts.** The most commonly encountered directed triangle is the "fan-in" motif, where two neurons converge on a single downstream target. **b. MICrONS Phase 1 directed triangle counts.** The most common motif is the "fan-out" motif, where one neuron synapses on two downstream targets.

3.5 Benchmarks

We compared the performance of three DotMotif executors available for use in our downloadable Python module. Results shown in **Figure 4** demonstrate that for graphs of $|E| > 10^4$, the `Neo4jExecutor` outperforms the `NetworkXExecutor` in query response-time. **Figure 5** illustrates that the *GrandIso*-based executor always outperforms the NetworkX executor on graphs that fit in memory.

Users may find that different executors suit the needs of different research questions. The *NetworkXExecutor* and *GrandIsoExecutor*, which convert DotMotif DSL syntax into a series of Python commands, are preferable for running graph queries on a small host graph, or for running the same queries on many host graphs, due to low startup-time overhead. The *Neo4jExecutor*, which converts DotMotif DSL syntax into a Cypher query for execution with the Neo4j graph database, has significant data-ingest overhead for the first query on a graph, but in general tends to be much faster than NetworkX for all subsequent queries. For queries on host-graphs of sufficient size, the *Neo4jExecutor* startup overhead becomes trivial compared to the total runtime of the query, and for larger-than-memory host-graphs, the *Neo4jExecutor* and *NeuPrintExecutors* may be the only feasible options. For most in-memory exploratory data analysis, we recommend the use of the *GrandIsoExecutor*.

The Neo4jExecutor has the dramatic advantage of caching graph data to disk. Users may find that for a sufficiently large graph, it is not possible to use executors that require the full query to run in RAM. In our experience, even when a user has access to a multi-terabyte RAM node, the Neo4j Java-based executor outperforms the Python-based NetworkX executor on large or high-density host graphs.

DotMotif users may trivially switch between Executors without modifying their queries, as the DotMotif engine stores queries in an intermediate format after parsing and validation steps (**Figure 6**).

All benchmarks listed here were performed on consumer laptop hardware with 16 gigabytes of RAM and a 3.1 GHz Intel Core i7 processor.

4 Methods

In developing DotMotif, we aimed to ensure that our software was both accessible — intuitive for new users but powerful enough for power-users — as well as sufficiently computationally efficient to execute most queries on commodity hardware.

In the interest of research flexibility, we also required that the software adapt to best utilize available hardware resources. For this reason, DotMotif includes several *executors*, each of which leverage a different graph analysis technology. Executors are discussed in greater depth in the *Benchmarks* section.

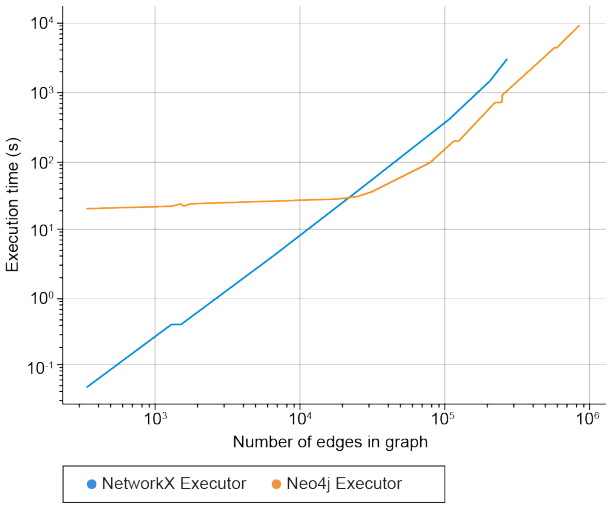

Figure 4: Comparison of executor runtime when counting undirected triangles. Here, two DotMotif *Executors* are compared counting triangle motifs in Erdős–Rényi random graphs. All times are reported from a cold start (i.e. the executor was not running prior to the query) and so all Neo4jExecutor results include start-up overhead that is not a factor for subsequent queries on the same host graph. Axes are on a log-log scale.

4.1 Formal definition of the subgraph search task

Here we consider the ideas of *subgraph isomorphism* and *subgraph monomorphism*, which have slightly different definitions in graph theory literature. When we refer to a *subgraph*, we always refer to a *node-induced* subgraph, which is inline with the usage in popular subgraph isomorphism research, including research on the commonly used VF2 algorithm [31, 32, 33].

Given graphs $G = (V_1, E_1)$ and $H = (V_2, E_2)$, we say G is isomorphic to H if there exists a bijection $f : V_1 \to V_2$ such that $\{u, v\} \subseteq E_1$ if and only if $\{f(u), f(v)\} \subseteq E_2$ [34]. An extension of this idea is that a subgraph isomorphism exists between G and H if and only if there exists a subgraph G' of G such that G' is isomorphic to H. A monomorphism loosens the bijection requirement to simply injection, meaning the search graph can contain extra edges.

DotMotif finds all subgraphs in a search graph that match a query graph, where the term *match* used here means that one of the above described mappings exists. By default, DotMotif will perform a search for all subgraphs that are monomorphic to the query subgraph. A user may choose to search for exact matches only, in which case DotMotif identifies subgraph *iso-*

morphisms, rather than *monomorphisms*. This behavior is user-toggleable during the construction of a query.

4.2 Optimizations to the subgraph isomorphism task in Python

For the purposes of quick iteration upon neuroscience hypotheses, our team required a high-speed subgraph isomorphism algorithm that was both pure Python — in order to reduce the barrier to entry for a fresh installation — as well as optimized to minimize CPU and RAM usage. To satisfy this need, we developed a subgraph monomorphism search implementation that performs favorably against the standard NetworkX implementation (Fig 5). Our implementation matches the results of the NetworkX GraphMatcher monomorphism detection module, with significantly lower CPU and memory overhead.

Additionally, while many modern subgraph iso- and monomorphism implementations rely upon a directed, acyclic graph state-space representation, our implementation stores its state space in a one-dimensional queue (**Algorithm 1**). In future work, we believe that due to this property, our implementation may be trivially parallelized to operate on multiple cores at once.

Algorithm 1: Our approach: subgraph isomorphism mapping

```
Input  : A small motif graph M
Input  : An arbitrarily large host graph H
Initialize an empty results set R;
Initialize an empty task queue Q;
Add the empty candidate mapping ({}) to Q;
while Q is non-empty do
    Pop a new candidate B from Q;
    Identify m_1 the most interesting node in
      motif M that does not yet have a mapping
      assigned in B;
    Identify all nodes that are valid mappings
      from the candidate to m_1 (based upon
      degree, attributes, etc);
    if multiple nodes are valid candidates then
        Add each new candidate to Q;
    else
        (All nodes in M have a valid mapping in
         B to H);
        Add the mapping to the results set R;
    end
end
Result: The set R of resultant mappings from
        nodes in M to IDs in H
```

4.3 Optimized interfaces for common graph libraries

Many existing research questions have been explored with RAM-sized graphs in mind, and have utilized popular graph libraries such as NetworkX or IGraph [15, 35], or emerging libraries such as Networkit [36]. In order to easily transition these algorithms and analyses to graphs of larger size, we developed an open-source graph library that enables a user to write code using familiar graph APIs (e.g. NetworkX's `nx.DiGraph`), but which automatically converts these commands to run instead on any of a number of optimized, scalable backends, such as a graph database implemented in SQL or Amazon Web Services DynamoDB. This tool, alongside the subgraph iso- and monomorphism improvements detailed above, enabled us to achieve substantial performance speed-ups in performing the analyses in this paper. Links to documentation and the source code for this library is available in *Supplemental Materials*.

4.4 Comparison to Random Graph Models

One way to determine if a motif is significant within a real biological dataset is to compare to its frequency within a random graph model. There are several different random graph models, each with different properties and capturing a different aspect of real world data. In this manuscript, we perform subgraph search on five different random graph models.

The Erdős–Rényi random model has parameters n, or the number of nodes, and p, or the probability that any of the potential $\binom{n}{2}$ edges will independently exist within the graph [22]. The Erdős–Rényi model is easy to analyze and understand and thus is included in this analysis. However, the edge independence assumption often fails for real datasets. Thus, other models are required to provide further context.

Geometric random graphs are desirable because they naturally capture spatial relationships that occur within the brain. Nodes are uniformly distributed within a region and two nodes are adjacent is they fall within a radius r, which is an adjustable parameter [23]. This model, however, does not capture the heavy-tailed distribution of edge weights across space, resulting in some spatial relationships that are not captured by the point cloud approach of the geometric model.

The Watts-Strogatz model addresses the edge dependence issue more directly, recognizing that many real world networks have high clustering coefficient, which is a measure of the extent to which two adjacent nodes have similar neighbors [24]. This phenomenon is called *preferential attachment*, and is seen in many real world datasets such as social networks or complete connectomes such as that of *C. elegans*.

The Barabási-Albert extends Watts-Strogatz to en-

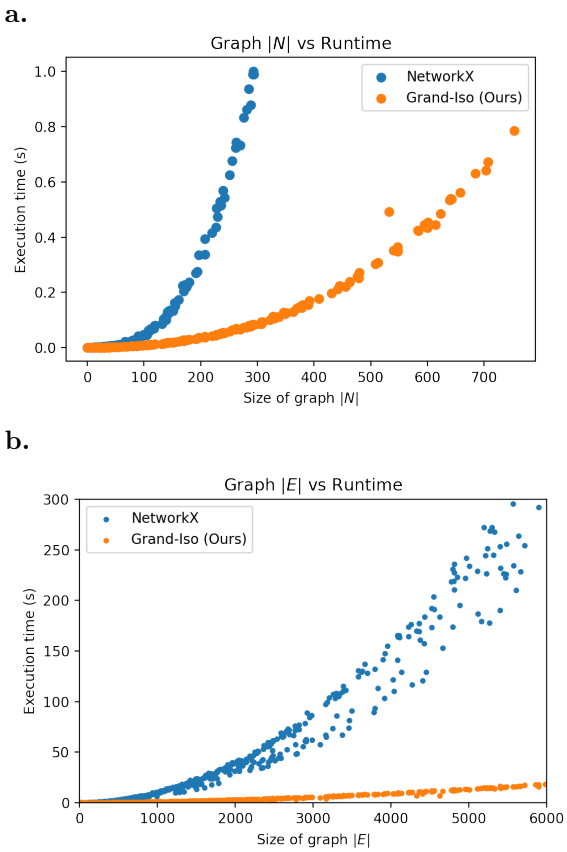

Figure 5: Comparison of wall-clock runtime when counting undirected motifs. All runtimes are measured during motif searches for rectangles in Erdős–Rényi graphs of various sizes. All graph density parameters are 0.15. **a. Execution time as a function of node count.** Experiments were terminated if they exceeded one second of runtime. **b. Execution time as a function of edge count.** All experiments were performed on otherwise-idle hardware.

sure the model also has a power law degree distribution, making it a scale free graph model [25].

The degree-preserving edge-randomization graph model implemented and used here, based upon efforts such as [26], accepts a graph as input-parameter and performs edge-swaps between two randomly-selected edges $\{u_1, v_1\}$ and $\{u_2, v_2\}$ so that the resulting edges are $\{u_1, v_2\}$ and $\{u_2, v_1\}$. This model preserves in- and out-degree distribution, as well as in-degree sequence.

Each of the described random models has a set of parameters resulting in different graph properties. These parameters may be selected to generate models that generate graphs "similar" to the real dataset in question. Similarity can mean different things and we try to match the models to a characteristic that is closely tied to motif search. One obvious choice is to match the graph density (D$=\frac{2|E|}{|V|(|V|-1)}$) of each estimated connectome because density is a metric that can be more easily tweaked across models. Furthermore, a density that is too high or too low can result in inflated or underestimated subgraph counts, respectively [37]. Other metrics besides graph density may be considered, such as degree distribution. Note that one implication is that the parameters selected for a random graph model will differ for each connectome being analyzed. More specifically, differences in density for *C. elegans* and MICrONS datasets will result in differently parametrized random graph models.

By performing subgraph search across these models, tuned or calibrated to some characteristic of the real dataset, we narrow the problem space of what drives a given motif to occur at a given frequency.

4.5 Architecture

DotMotif is comprised of three submodules: A `parser` module, an `optimizer` module, and finally an `executor` module. In order of use, the `parser` module is responsible for converting the DotMotif domain-specific language into an in-memory representation. The `optimizer` module is then responsible for converting and simplifying the in-memory motif into its simplest possible representation. The `optimizer` module may also optionally check for violations of biological priors, in a validation step. Finally, the `executor` module converts the optimized motif into a query that can be submitted to a graph analysis tool. Each executor is responsible for generating its own target-specific queries. (For example, the `Neo4jExecutor` generates Cypher queries, and the `NetworkXExecutor` converts queries to a sequence of Python commands.)

4.5.1 DotMotif Domain-Specific Language Parser

We identified a disparity between the flexibility of common query languages (such as Cypher [5]) and the needs of the research community, where many research questions require overwhelmingly complex or verbose queries. We opted to develop a domain-specific query language to aid in the construction of queries. This enables research-driven query design agnostic to the underlying frameworks.

The DotMotif domain-specific language (DSL) borrows from DOT-like syntax [38] as well as from SQL-like syntax in order to expose a succinct and user-friendly query language.

For example, the simple query `A -> B` will return a list of all edges in the complete graph (in other words, the list of all subgraphs of $G = (V, E)$ that comprise one edge from a node $A \in V$ to node $B \in V$).

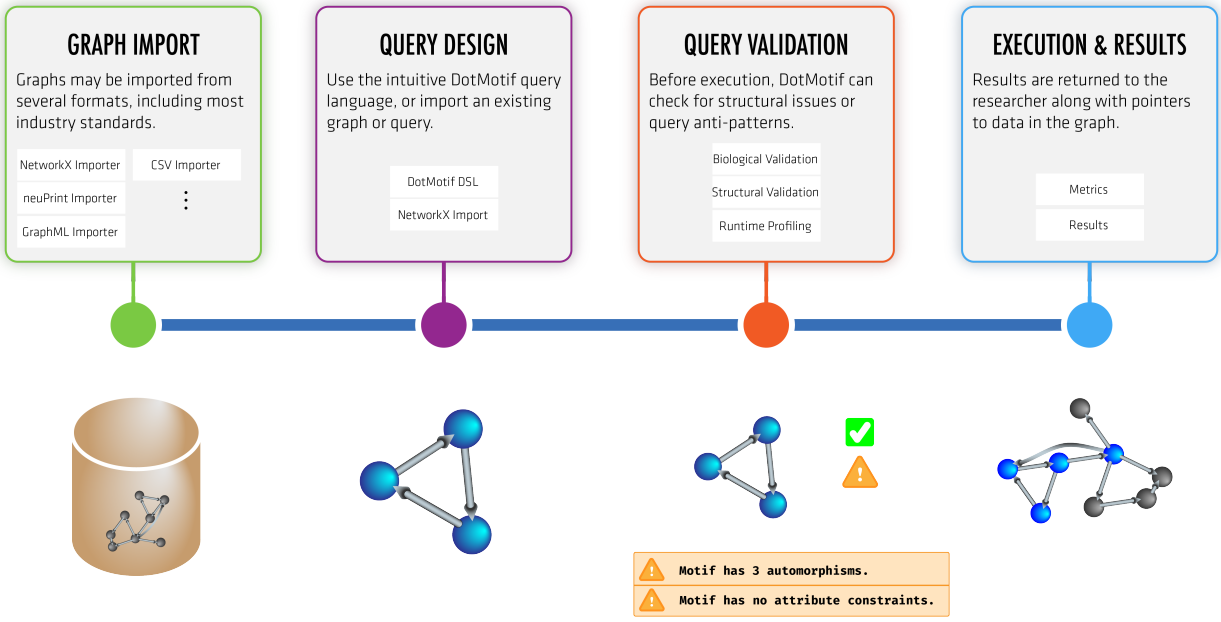

Figure 6: The DotMotif query execution process.
Graph Import. A user may import a search graph (also commonly referred to as a *host* graph) from a variety of industry standard formats, including edgelist CSV, GraphML, numpy adjacency matrix, and any format supported by the NetworkX library. Furthermore, DotMotif is compatible with graphs stored in neuPrint schemas in a Neo4j database [4]. **Query Design.** The user may select from a library of pre-built motifs, or write a novel query. The query may be written in the DotMotif DSL, encoded as a NetworkX graph, or written as text directly in the target graph database query language, such as Cypher. A user may choose to save this query as another standalone file on disk for future use. The .motif file-format stores both the motif itself as well as provenance metadata such as the date of creation, comments associated with the motif, and author information. **Query Validation.** DotMotif supports several pre-execution validation steps in order to fail quickly when asked to perform an impossible or paradoxical query. Several other optional validators may be invoked to check a motif for biological feasibility and warn the user if a potential error is detected. **Execution & Results.** The user may choose from the available DotMotif Executors in order to use the most appropriate subgraph matching tool for the compute resources available. No further query modification is required in order to run the same motif on several different Executors. A user may also choose to simply use DotMotif as a query executor without leveraging the query validation or query design tools.

A query of `A -> B; B -> C; C -> A` will return a list of all triangles with edges of the same direction in the complete graph. A query of `A -> B; B !> A` will return a list of all edges in the graph for which the reciprocal edge does not exist.

In order to make queries understandable and maintainable, the DotMotif DSL supports "macros," or composable building-blocks that can be combined to generate more complicated queries. For example, if we wish to identify all inhibitory neurons with an annotated radius of 10 units or more, with a GABAergic synapse onto a node with radius of 5 units or smaller, we can write the following query:

```
Big -> Small [type=GABA]
Big.radius >= 10
Small.radius <= 5
```

If we want to extend this query to find all cases where *two* 'Big' neurons synapse on the same 'Small' neuron, we can use a macro (DotMotif comments are preceded by a `#` character):

```
BigInhibitsSmall(Big, Small) {
  # A macro for a single synapse
  # under investigation
  Big -> Small [type=GABA]
  Big.radius >= 10
  Small.radius <= 5
}

# Search for two of this pattern
# that share a postsynaptic partner:
BigInhibitsSmall(A, C)
BigInhibitsSmall(B, C)
```

These macros minimize "boilerplate" syntax without leading to duplicate notation. By default, Dot-

Motif returns a single representative element of each automorphism group in the result set. An *automorphism* is an isomorphism from a graph onto itself [34]. Returning a single representative of the automorphism group avoids overcounting motifs in the search graph. For example, if a user requests a triangle:

```
A -> B
B -> C
C -> A
```

...then there are three possible automorphisms for each node A, B, and C (the same set of three nodes in the host graph may be assigned to any of the nodes in the motif query). In this case, DotMotif will choose one arbitrarily. In this counter-example,

```
A -> B
B -> C
C -> A
A.brainArea != "CX"
```

...there are no motif automorphisms because the node-attribute constraint on A makes it unique from the unconstrained nodes B and C.

A user may explicitly specify that two nodes in a motif are isomorphic with the `===` notation:

```
A -> B
B -> C
C -> A

A === B
B === C
```

The DotMotif `===` automorphism operator is transitive and symmetric, so `A===C` is unnecessary in this query.

If the user uses the automatic query automorphism detection (the former example) or notates it manually (the latter example), the DotMotif optimizer will enrich the query motif to lower the space of possible matches, and thus return a result more rapidly. DotMotif considers both semantic as well as syntactic feasibility when automatically determining automorphism mappings [32].

This DSL is formally defined in Extended Backus–Naur Form [39, 40] by the grammar file referenced in *Supplemental Materials*. This syntax is visualized using the railroad-diagram convention in **Figure 7**.

4.5.2 Query Optimization and Validation

After a query has been parsed and ingested, it is passed to an optimization stage in which the query is reduced to a simplified form in order to improve the speed of query execution. Optionally, a user may also enable the automatic detection of impossible structures — or even biologically implausible structures — in a *Validation* step.

For example, the following simple *.motif* file would fail structural validation, because it calls for an edge between nodes A and B to both exist as well as *not* exist:

```
A -> B
A !> B
```

This motif fails with the following error message:

```
DisagreeingEdgesValidatorError:
  Trying to add <[A]-[B] exists=False> but
  <[A]-[B] exists=True> already in motif.
```

This second example fails validation due to the impossible requirements placed on the size of the A neuron:

```
A.size > 50
A -> B
A.size <= 25

DisagreeingNodeConstraintError:
  Trying to add ([A].size <= 25)
  but constraint ([A].size > 50)
  already in motif.
```

These verbose error messages are intended to warn a user quickly and clearly if a graph query is unlikely or impossible, rather than allowing the user to proceed with a long-running query that is destined to fail. In contrast, the equivalent Cypher command running in Neo4j will still go through the process of execution on the full host graph, but will return no results.

DotMotif includes the option to include validators for *biological plausibility*. These allow the user to prohibit motifs that — for example — require inhibition and excitation to be performed by the same neuron. These validators, like all validators in the DotMotif package, are both optional as well as modifiable.

4.5.3 Query execution

A parsed, optimized, and validated query can now be evaluated against a large graph. Where necessary, it is possible to run these queries fully in-memory on consumer hardware. DotMotif includes a `NetworkXExecutor`, which can search a graph for motifs in a pure Python implementation. This is ideal for very small graphs (see *Benchmarks*, below) or for constrained compute environments where more efficient executors cannot be used.

We have also implemented a `Neo4jExecutor`, which leverages the Neo4j [5] database and its built-in subgraph match detection algorithm, an implementation

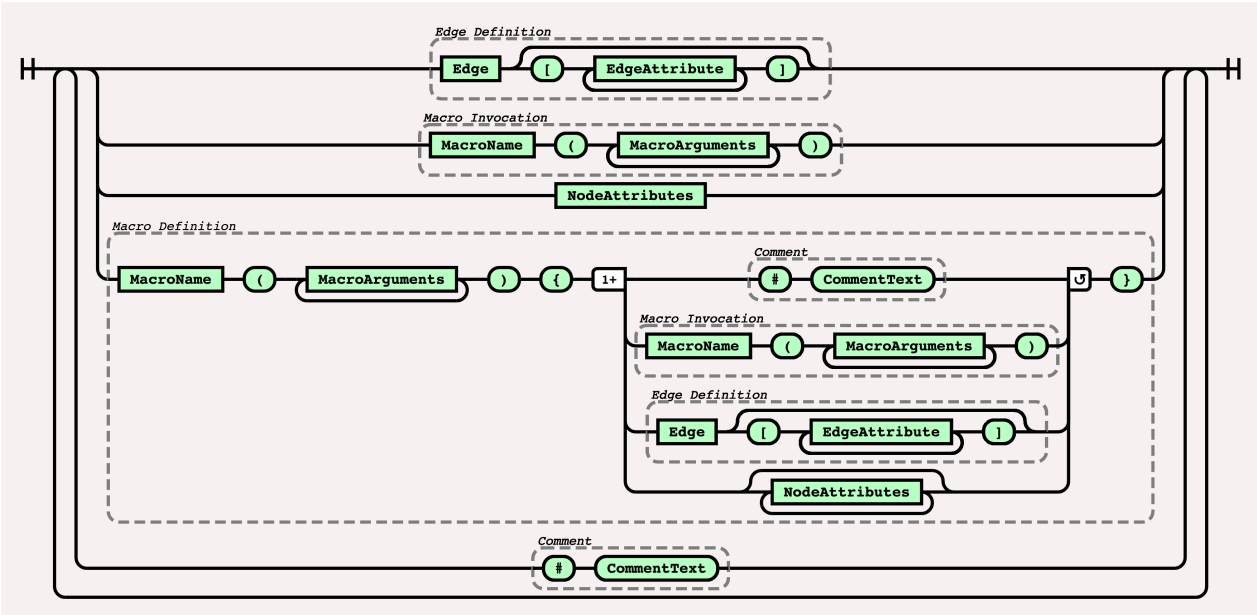

Figure 7: The DotMotif Syntax in railroad-diagram form. The DotMotif DSL is a whitespace-agnostic language with hash-symbol comments and curly-brace bracketed macros. Edges are defined with `->`-arrowlike syntax; edge attributes may be listed in JSON-like syntax. Node attributes may be listed in object dot-notation syntax. For example usage of this syntax system, refer to *Architecture*. This syntax is formally defined in Extended Backus-Naur Form. Both this formal definition as well as the language specification will be made available as per *Supplemental Materials*. This figure was generated with the *railroad-diagrams* Node.js package [41].

of the *VF2* algorithm [31, 32]. In order to run queries in this environment, our software converts the optimized query to *Cypher*, the database query language used natively by Neo4j. If a user does not have a running Neo4j database, our library also includes routines to provision a Docker container [42] and quickly ingest the data in an appropriate format for Neo4j to use.

5 Discussion

Modern connectomics research currently suffers from a lack of accessible, yet performant, graph analysis and subgraph-search tools. In this work, we present DotMotif, a combined domain-specific language and interface to powerful graph-search tools. Our hope is that this tool and others like it will reduce the barrier-to-entry for researchers unacquainted with graph theory or graph databases, and will enable researchers to interrogate increasingly-common connectome datasets with ease.

Though the undirected graphs found in the graph atlas study here may not point to specific results that directly illuminate neuroscience answers, this broad search can narrow down where it may be most impactful to dig deeper, perhaps leading to more interesting directed graph searches or neural simulations over smaller directed version. Future work may include random direction assignment in order to perform an unbiased search across the space of all possible graphs, akin to other random-graph searches [6].

It is interesting to note that the MICrONS dataset closely matched all random graph models but the geometric random graph. Indeed, the Watts-Strogatz graph model, which greatly underestimated motif counts when calibrated to match the *C. elegans* graph, was a much better predictor of motif counts in the MICrONS dataset. We intend to further explore other metrics beyond density for random graph model calibration in the future.

Despite many of the simplifications and optimizations to the subgraph monomorphism task mentioned here, this task in general is NP-complete [43, 33, 44, 45]. Even in a relatively small connectome such as that of *C. elegans*, certain sufficiently common or poorly-optimized motif searches may still remain infeasible to run on consumer hardware. In our experience, the addition of further constraints, such as specifying edge direction, specifying node- and edge-attributes, or searching for query graphs with nodes of high degree, may reduce the execution time of complex queries; but other seemingly helpful modifications, such as increasing the diameter of a query graph, may counter-intuitively increase runtime quite substantially. As a result of such longer query run-

times or more compute-hungry queries, researchers will be able to run fewer queries per study.

In our ongoing work, we hope to help to reduce this barrier by developing more rigorous automated query optimizations and by publishing connectome query results for reuse. We will introduce and refine new validators that aid DotMotif query designers in identifying qualities of a query graph that may lead to long runtimes, and we hope that the atlas results published above may shed light on how to select query graphs for further study in a dataset. We intend to disseminate an open-data *motif encyclopedia* so that long-running or complex queries may be run on a dataset once and then shared with the community.

We also note that faster implementations of the motif-search problem — relevant to connectomics, pharmaceutical research, and other domains — may pave the way to faster and more efficient scientific discovery.

The field of connectomics is at an inflection point, as technology and neuroscience provide avenues to create and study large datasets at unprecedented scales beyond the analysis capabilities of a single lab. Open, free, and public datasets, as well as accessible and affordable tools to understand those public data, are of paramount importance.

We are releasing the DotMotif codebase, as well as all demonstration code and data in this manuscript, as open-source and open-data tools to support community discovery. We invite collaborators to share questions that allow us extend DotMotif to test scientific hypotheses.

6 Supplemental Materials

Code for the DotMotif Python library, the example snippets in this paper, and all data and code for the experiments run in *Results* are available online at *https://bossdb.org/tools/dotmotif*.

Code and documentation for the improved subgraph isomorphism library is available online at *https://bossdb.org/tools/grandiso*, and the graph-library abstraction tool discussed in *Optimized interfaces for common graph libraries* is available at *https://bossdb.org/tools/grand*.

7 Acknowledgements

The authors would like to thank Jennifer Stiso, Danielle Bassett, and Konrad Kording for advice and review of this manuscript.

Research reported in this publication was supported by the National Institute of Mental Health of the National Institutes of Health under Award Numbers R24MH114799 and R24MH114785. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. This work was also completed with the support of JHU/APL Internal Research Funding.

We would like to thank the teams that generated the connectome data used in this analysis.

References

[1] C. S. Xu, M. Januszewski, Z. Lu, S.-y. Takemura, K. J. Hayworth, G. Huang, K. Shinomiya, J. Maitin-Shepard, D. Ackerman, S. Berg, and et al., "A connectome of the adult drosophila central brain," Jan 2020. [Online]. Available: http://dx.doi.org/10.1101/2020.01.21.911859

[2] R. Burns, E. Perlman, A. Baden, W. G. Roncal, B. Falk, V. Chandrashekhar, F. Collman, S. Seshamani, J. Patsolic, and K. Lillaney, "A Community-Developed Open-Source Computational Ecosystem for Big Neuro Data," *arXiv preprint arXiv:1804.02835*, 2018.

[3] T. Hočevar and J. Demšar, "Combinatorial algorithm for counting small induced graphs and orbits," *PLOS ONE*, vol. 12, no. 2, p. e0171428, Feb 2017. [Online]. Available: http://dx.doi.org/10.1371/journal.pone.0171428

[4] J. Clements, T. Dolafi, L. Umayam, N. L. Neubarth, S. Berg, L. K. Scheffer, and S. M. Plaza, "neuprint: Analysis tools for em connectomics," Jan 2020. [Online]. Available: http://dx.doi.org/10.1101/2020.01.16.909465

[5] Neo4j.org, "Neo4j Graph Platform." [Online]. Available: https://neo4j.com

[6] O. Sporns and R. Kötter, "Motifs in brain networks," *PLOS BIOL*, p. 369, 2004.

[7] V. B. Mountcastle, "Modality and topographic properties of single neurons of cat's somatic sensory cortex," *Journal of Neurophysiology*, vol. 20, no. 4, p. 408–434, Jul 1957. [Online]. Available: http://dx.doi.org/10.1152/jn.1957.20.4.408

[8] N. L. Turner, T. Macrina, J. A. Bae, R. Yang, A. M. Wilson, C. Schneider-Mizell, K. Lee, R. Lu, J. Wu, A. L. Bodor, and et al., "Multiscale and multimodal reconstruction of cortical structure and function," Oct 2020. [Online]. Available: http://dx.doi.org/10.1101/2020.10.14.338681

[9] N. Pospelov, S. Nechaev, K. Anokhin, O. Valba, V. Avetisov, and A. Gorsky, "Spectral peculiarity and criticality of a human connectome," *Physics of Life Reviews*, vol. 31, p. 240–256, Dec 2019. [Online]. Available: http://dx.doi.org/10.1016/j.plrev.2019.07.003

[10] E. P. Reilly, J. S. Garretson, W. R. Gray Roncal, D. M. Kleissas, B. A. Wester, M. A. Chevillet, and M. J. Roos, "Neural reconstruction integrity: A metric for assessing the connectivity accuracy of reconstructed neural networks," *Frontiers in neuroinformatics*, vol. 12, p. 74, 2018.

[11] E. P. Reilly, E. C. Johnson, M. J. Hughes, D. Ramsden, L. Park, B. Wester, and W. Gray-Roncal, "Connecting neural reconstruction integrity (nri) to graph metrics and biological priors," in *Complex Networks XI*. Springer, 2020, pp. 182–193.

[12] O. Sporns, "Structure and function of complex brain networks," *Dialogues in Clinical Neuroscience*, vol. 15, pp. 247 – 262, 2013.

[13] L. W. Swanson and J. W. Lichtman, "From cajal to connectome and beyond," *Annual Review of Neuroscience*, vol. 39, no. 1, p. 197–216, Jul 2016. [Online]. Available: http://dx.doi.org/10.1146/annurev-neuro-071714-033954

[14] D. S. Bassett and O. Sporns, "Network neuroscience," *Nature Neuroscience*, vol. 20, no. 3, p. 353–364, Feb 2017. [Online]. Available: http://dx.doi.org/10.1038/nn.4502

[15] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[16] "Cayley." [Online]. Available: https://cayley.io/

[17] T. A. Jarrell, Y. Wang, A. E. Bloniarz, C. A. Brittin, M. Xu, J. N. Thomson, D. G. Albertson, D. H. Hall, and S. W. Emmons, "The connectome of a decision-making neural network," *Science*, vol. 337, no. 6093, p. 437–444, Jul 2012. [Online]. Available: http://dx.doi.org/10.1126/science.1221762

[18] S. J. Cook, T. A. Jarrell, C. A. Brittin, Y. Wang, A. E. Bloniarz, M. A. Yakovlev, K. C. Q. Nguyen, L. T.-H. Tang, E. A. Bayer, J. S. Duerr, and et al., "Whole-animal connectomes of both caenorhabditis elegans sexes," *Nature*, vol. 571, no. 7763, p. 63–71, Jul 2019. [Online]. Available: http://dx.doi.org/10.1038/s41586-019-1352-7

[19] S. Dorkenwald, N. L. Turner, T. Macrina, K. Lee, R. Lu, J. Wu, A. L. Bodor, A. A. Bleckert, D. Brittain, N. Kemnitz, and et al., "Binary and analog variation of synapses between cortical pyramidal neurons," Dec 2019. [Online]. Available: http://dx.doi.org/10.1101/2019.12.29.890319

[20] C. M. Schneider-Mizell, A. L. Bodor, F. Collman, D. Brittain, A. A. Bleckert, S. Dorkenwald, N. L. Turner, T. Macrina, K. Lee, R. Lu, and et al., "Chandelier cell anatomy and function reveal a variably distributed but common signal," Apr 2020. [Online]. Available: http://dx.doi.org/10.1101/2020.03.31.018952

[21] R. C. Read and R. J. Wilson, *An Atlas of Graphs*. USA: Oxford University Press, Inc., 2005.

[22] P. Erdös and A. Rényi, "On random graphs i," *Publicationes Mathematicae Debrecen*, vol. 6, p. 290, 1959.

[23] D. Penrose, M. Penrose, and O. U. Press, *Random Geometric Graphs*, ser. Oxford studies in probability. Oxford University Press, 2003. [Online]. Available: https://books.google.com/books?id=RHvnCwAAQBAJ

[24] D. J. Watts and S. H. Strogatz, "Collective dynamics of "small-world" networks," *Nature*, vol. 393, no. 6684, p. 440–442, Jun 1998. [Online]. Available: http://dx.doi.org/10.1038/30918

[25] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of Modern Physics*, vol. 74, no. 1, p. 47–97, Jan 2002. [Online]. Available: http://dx.doi.org/10.1103/RevModPhys.74.47

[26] E. S. Roberts and A. C. C. Coolen, "Unbiased degree-preserving randomisation of directed binary networks," 2011.

[27] J. White, E. Southgate, J. N. Thomson, and S. Brenner, "The structure of the nervous system of the nematode C. elegans," *Philosophical transactions Royal Society London*, vol. 314, pp. 1–340, 1986.

[28] C. A. Brittin, S. J. Cook, D. H. Hall, S. W. Emmons, and N. Cohen, "Volumetric reconstruction of main caenorhabditis elegans neuropil at two different time points," Dec 2018. [Online]. Available: http://dx.doi.org/10.1101/485771

[29] E. Gal, R. Perin, H. Markram, M. London, and I. Segev, "Neuron geometry underlies universal network features in cortical microcircuits," May

2019. [Online]. Available: http://dx.doi.org/10.1101/656058

[30] C. Curto, C. Langdon, and K. Morrison, "Robust motifs of threshold-linear networks," 2019.

[31] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "An improved algorithm for matching large graphs," 2001.

[32] L. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, pp. 1367 – 1372, 11 2004.

[33] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM (JACM)*, vol. 23, no. 1, p. 31–42, Jan 1976. [Online]. Available: http://dx.doi.org/10.1145/321921.321925

[34] D. B. West *et al.*, *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2.

[35] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal*, vol. Complex Systems, p. 1695, 2006. [Online]. Available: http://igraph.org

[36] C. L. Staudt, A. Sazonovs, and H. Meyerhenke, "Networkit: A tool suite for large-scale complex network analysis," 2015.

[37] B. C. M. van Wijk, C. J. Stam, and A. Daffertshofer, "Comparing brain networks of different size and connectivity density using graph theory," *PLoS ONE*, vol. 5, no. 10, p. e13701, Oct 2010. [Online]. Available: http://dx.doi.org/10.1371/journal.pone.0013701

[38] E. R. Gansner, E. Koutsofios, S. C. North, and K. phong Vo, "A technique for drawing directed graphs," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 19, no. 3, pp. 214–230, 1993.

[39] J. W. Backus, "The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference." in *IFIP Congress*. Butterworths, London, 1959, pp. 125–131. [Online]. Available: http://dblp.uni-trier.de/db/conf/ifip/ifip1959.html#Backus59

[40] D. E. Knuth, "Backus Normal Form vs. Backus Naur Form," *Communications of the ACM*, vol. 7, no. 12, p. 735–736, Dec 1964. [Online]. Available: http://dx.doi.org/10.1145/355588.365140

[41] T. Atkins-Bittner, "Railroad-diagram generators," https://github.com/tabatkins/railroad-diagrams, 2020.

[42] "Docker." [Online]. Available: https://www.docker.com/index.html

[43] M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han, "Efficient subgraph matching," *Proceedings of the 2019 International Conference on Management of Data - SIGMOD '19*, 2019. [Online]. Available: http://dx.doi.org/10.1145/3299869.3319880

[44] V. Vassilevska and R. Williams, "Finding, minimizing, and counting weighted subgraphs," *Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09*, 2009. [Online]. Available: http://dx.doi.org/10.1145/1536414.1536477

[45] P. Hell and J. Nešetřil, "Colouring, constraint satisfaction, and complexity," *Computer Science Review*, vol. 2, no. 3, p. 143–163, Dec 2008. [Online]. Available: http://dx.doi.org/10.1016/j.cosrev.2008.10.003