Hetnet connectivity search provides rapid insights into how two biomedical entities are related

This manuscript (permalink) was automatically generated from greenelab/connectivity-search-manuscript@401f8b7 on September 2, 2021.

Authors

Manuscript in preparation
The authorship information below is incomplete and preliminary. This notice will be updated once all contributors meeting authorship criteria have added themselves to `metadata.yaml`.

- **Daniel S. Himmelstein**
  0000-0002-3012-7446 · dhimmel · dhimmel
  Department of Systems Pharmacology and Translational Therapeutics, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America; Related Sciences · Funded by GBMF4552

- **Michael Zietz**
  0000-0003-0539-630X · zietzm · ZietzMichael
  Department of Systems Pharmacology and Translational Therapeutics, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America; Department of Biomedical Informatics, Columbia University, New York, New York, United States of America

- **David N. Nicholson**
  0000-0003-0002-5761 · danich1
  Department of Systems Pharmacology and Translational Therapeutics, Perelman School of Medicine University of Pennsylvania, Philadelphia PA, USA · Funded by The Gordon and Betty Moore Foundation (GBMF4552); The National Institutes of Health (T32 HG000046)

- **Casey S. Greene**
  0000-0001-8713-9213 · cgreene · GreeneScientist
  Department of Systems Pharmacology and Translational Therapeutics, University of Pennsylvania, Philadelphia, Pennsylvania, United States of America; Department of Biochemistry and Molecular Genetics, University of Colorado School of Medicine, Aurora, Colorado, United States of America; Center for Health AI, University of Colorado School of Medicine, Aurora, Colorado, United States of America · Funded by the National Human Genome Research Institute (R01 HG010067); the National Cancer Institute (R01 CA237170); the Gordon and Betty Moore Foundation (GBMF 4552)

Abstract

Hetnets, short for “heterogeneous networks”, contain multiple node and relationship types and offer a way to encode biomedical knowledge. For example, Hetionet connects 11 types of nodes — including genes, diseases, drugs, pathways, and anatomical structures — with over 2 million edges of 24 types. Previously, we trained a classifier to repurpose drugs using features extracted from Hetionet. The model identified types of paths between a drug and disease that occurred more frequently between known treatments.

For many applications however, a training set of known relationships does not exist; Yet researchers would still like to learn how two nodes are meaningfully connected. For example, users may be curious not only how metformin is related to breast cancer, but also how the *GJA1* gene might be involved in insomnia. Therefore, we developed hetnet connectivity search to propose the most important paths between any two nodes.

The algorithm behind connectivity search identifies types of paths that occur more frequently than would be expected by chance (based on node degree alone). We implemented the method on Hetionet and provide an online interface at https://het.io/search. Several optimizations were required to precompute significant instances of node connectivity at scale. We provide an open source implementation of these methods in our new Python package named hetmatpy.

To validate the method, we show that it identifies much of the same evidence for specific instances of drug repurposing as the previous supervised approach, but without requiring a training set.

Introduction

A *network* (also known as a graph) is a conceptual representation of a group of entities — called *nodes* — and the relationships between them — called *edges*. Typically, a network has only one type of node and one type of edge. But in many cases, it is necessary to be able to distinguish between different types of entities and relationships.

Hetnets

A *hetnet* (short for **het**erogeneous information **net**work [1]) is a network where nodes and edges have type. The ability to differentiate between different types of entities and relationships allows a hetnet to accurately describe more complex data. Hetnets are particularly useful in biomedicine, where it is important to capture the conceptual distinctions between various concepts, such as genes and diseases, or upregulation and binding.

The types of nodes and edges in a hetnet are defined by a schema, referred to as a metagraph. The metagraph consists of metanodes (types of nodes) and metaedges (types of edges). Note that the prefix *meta* is used to refer to type (e.g. compound), as opposed to a specific node/edge/path itself (e.g. acetaminophen).

Hetionet

Hetionet is a knowledge graph of human biology, disease, and medicine, integrating information from millions of studies and decades of research. Hetionet v1.0 combines information from 29 public databases. The network contains 47,031 nodes of 11 types (Table 1) and 2,250,197 edges of 24 types (Figure 1A).

Table 1: Node types in Hetionet The abbreviation, number of nodes, and description for each of the 11 metanodes in Hetionet v1.0.

Metanode	Abbr	Nodes	Description
Anatomy	A	402	Anatomical structures, excluding structures that are known not to be found in humans. From Uberon.
Biological Process	BP	11381	Larger processes or biological programs accomplished by multiple molecular activities. From Gene Ontology.
Cellular Component	CC	1391	The locations relative to cellular structures in which a gene product performs a function. From Gene Ontology.
Compound	C	1552	Approved small molecule compounds with documented chemical structures. From DrugBank.

Metanode	Abbr	Nodes	Description
Disease	D	137	Complex diseases, selected to be distinct and specific enough to be clinically relevant yet general enough to be well annotated. From Disease Ontology.
Gene	G	20945	Protein-coding human genes. From Entrez Gene.
Molecular Function	MF	2884	Activities that occur at the molecular level, such as “catalysis” or “transport”. From Gene Ontology.
Pathway	PW	1822	A series of actions among molecules in a cell that leads to a certain product or change in the cell. From WikiPathways, Reactome, and Pathway Interaction Database.
Pharmacologic Class	PC	345	“Chemical/Ingredient”, “Mechanism of Action”, and “Physiologic Effect” FDA class types. From DrugCentral.
Side Effect	SE	5734	Adverse drug reactions. From SIDER/UMLS.
Symptom	S	438	Signs and Symptoms (i.e. clinical abnormalities that can indicate a medical condition). From the MeSH ontology.

Hetionet provides a foundation for building hetnet applications. It unifies data from several different, disparate sources into a single, comprehensive, accessible, common-format network. The database is publicly accessible without login at https://neo4j.het.io. The Neo4j graph database enables querying Hetionet using the Cypher language, which was designed to interact with networks where nodes and edges have both types and properties.

One limitation that restricts the applicability of Hetionet is incompleteness. In many cases, Hetionet v1.0 includes only a subset of the nodes from a given resource. For example, the Disease Ontology contains over 9,000 diseases [2], while Hetionet includes only 137 diseases [3]. Nodes were excluded to avoid redundant or overly specific nodes, while ensuring a minimum level of connectivity for compounds and diseases. See the Project Rephetio methods for more details [4]. Nonetheless, Hetionet v1.0 remains one of the most comprehensive and integrative networks that consolidates biomedical knowledge into a manageable number of node and edge types. Other integrative resources, some still under development, include Wikidata [5], SemMedDB [6,7,8], SPOKE, and DRKG.

Rephetio

Project Rephetio is the name of the study that created Hetionet and applied it repurpose drugs [4]. This project predicted the probability of drug efficacy for 209,168 compound–disease pairs. The approach learned which types of paths occur more or less frequently between known treatments

than non-treatments (Figure 1B). To train the model, Rephetio created PharmacotherapyDB, a physician-curated catalog of 755 disease-modifying treatments [9].

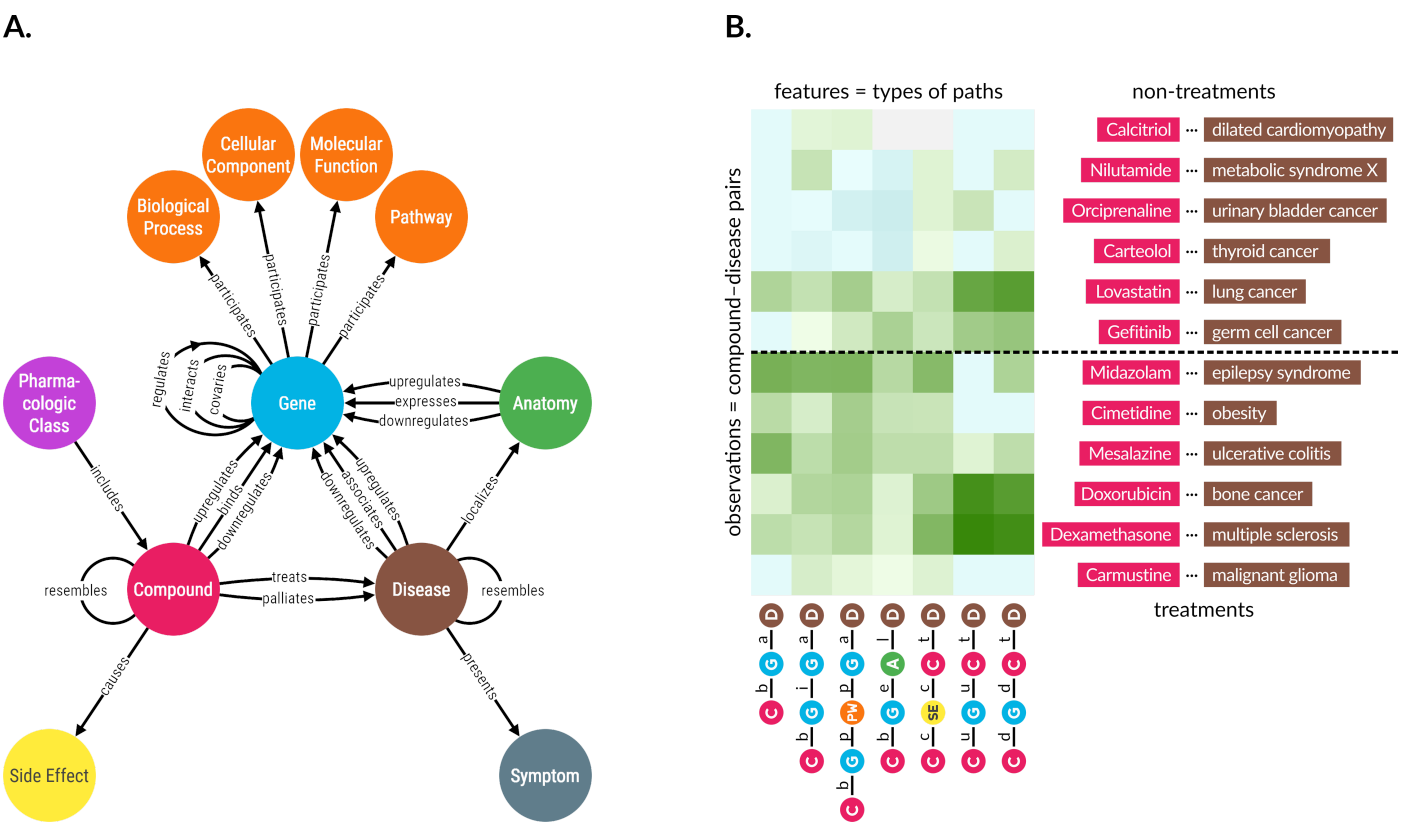

Figure 1: A. Hetionet v1.0 metagraph. The types of nodes and edges in Hetionet.
B. Supervised machine learning approach from Project Rephetio. This figure visualizes the feature matrix used by Project Rephetio to make supervised predictions. Each row represents a compound–disease pair. The top half of rows correspond to known treatments (i.e. positives), while the bottom half correspond to non-treatments (i.e. negatives under a *closed-world assumption*, not known to be treatments in PharmacotherapyDB). Here, an equal number of treatments and non-treatments are shown, but in reality the problem is heavily imbalanced. Project Rephetio scaled models to assume a positive prevalence of 0.36% [4,10]. Each column represents a metapath, labeled with its abbreviation.
Feature values are DWPCs (transformed and standardized), which assess the connectivity along the specified metapath between the specific compound and disease. Green colored values indicate above-average connectivity, whereas blue values indicate below average connectivity. In general, positives have greater connectivity for the selected metapaths than negatives. Rephetio used a logistic regression model to learn the effect of each type of connectivity (feature) on the likelihood that a compound treats a disease. The model predicts whether a compound–disease pair is a treatment based on its features, but requires supervision in the form of known treatments.

Unsupervised hetnet connectivity search

Project Rephetio was able to successfully predict treatments, including those under investigation by clinical trail. However, two challenges limit the applicability of the Rephetio approach, which this study aims to address. First, Rephetio required known labels (i.e. treatment status) to train a model. Hence, the approach cannot be applied to domains where training labels do not exist. Second, the DWPC metric used to assess connectivity is sensitive to node degree. The Rephetio approach was incapable of detecting whether a high DWPC score indicated meaningful connectivity above the level expected by the background network degrees. Here we propose Hetnet connectivity search, which defines a null distribution for DWPCs that accounts for degree and enables detecting meaningful hetnet connectivity without training labels.

Related Works

Existing research provides methods for determining whether two nodes are related, although primarily focuses on homogeneous networks (without type). Early approaches detected related nodes by measuring neighborhood overlap or path similarity between two nodes [11,12]. These approaches predicted node relatedness with success. However, they are difficult to scale as a network grows in size or semantic richness (i.e. type) [11].

More recently, focus has shifted to graph embeddings to determine if two nodes are related, specifically in the context of knowledge graphs, which are often semantically rich and include type [13,14,15,16,17]. These types of methods involve mapping nodes and sometimes edges to dense vectors via a neural network model [18,19,20], matrix factorization [21,22], or by translational distance models [23]. Once these dense vectors have been produced, quantitative scores that measure node relatedness can be generated via a machine learning model [14,24,25] or by selected similarity metrics [13,15,26,27,28]. These approaches have been quite successful in determining node relatedness. Yet, they only state *whether* two nodes are related and fail to provide an explanation on *why* two nodes are related.

Explaining why two nodes are related is a non-trivial task because approaches are required to output more information than a simple similarity score. The first group of approaches output a list of ranked paths that are most relevant between two nodes [29,30,31]. For example, the FAIRY framework explains for why items appear on a user's social media feed based on a network of users and content classes (e.g. categories, user posts, songs) [30]. ESPRESSO explains how two sets of nodes are related by returning subgraphs [32]. Other approaches such as MetaExp return important metapaths rather than paths, but require some form of supervision [33,34]. Our goal with Hetnet connectivity search is to explain how two nodes are related, while doing so in an unsupervised manner that captures the semantic richness of edge type and returns results in the form of both metapaths and paths.

Results

Connectivity Search Webapp

We created the connectivity search webapp available at https://het.io/search/. The tool is free to use, without any login or authentication. The purpose is let users quickly explore how any two nodes in Hetionet v1.0 might be related. The workflow is based around showing the user the most important metapaths and paths for a pair of query nodes.

The design guides the user through selecting a source and target node (Figure 2A). The webapp returns metapaths, scored by whether they occurred more than expected based on network degree (Figure 2B). Users can proceed by requesting the specific paths for each metapath, which are placed in a unified table sorted according to their path score (Figure 2C). Finally, the webapp produces publication-ready visualizations containing user-selected paths (Figure 2D).

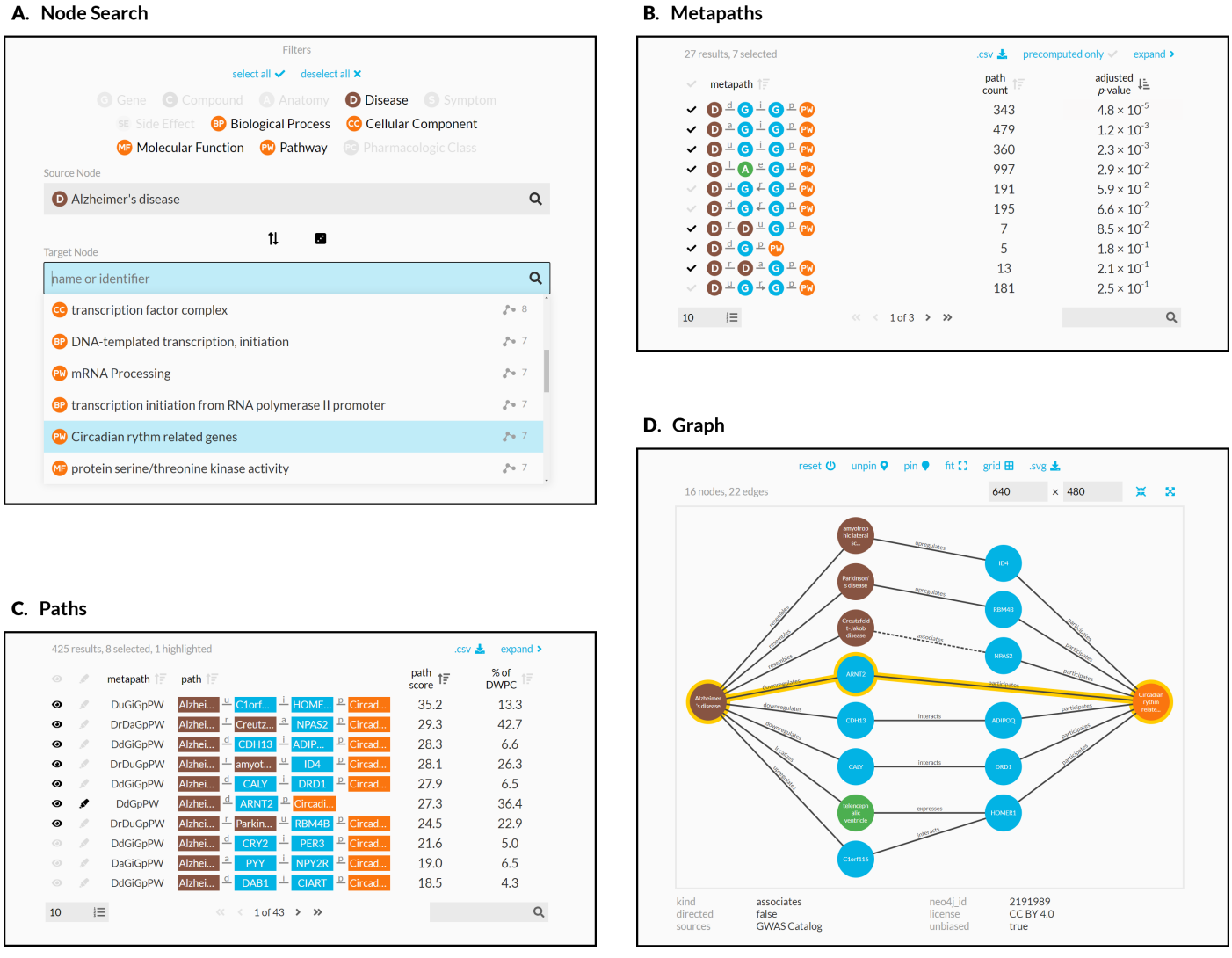

Figure 2: Using the connectivity search webapp to explore the pathophysiology of Alzheimer's disease. This figure shows an example user workflow for https://het.io/search/.
A. The user selects two nodes. Here, the user is interested in Alzheimer's disease, so selects this as the source node. The user limits the target node search to metanodes relating to gene function. The target node search box suggests nodes, sorted by the number of significant metapaths. When the user types in the target node box, the matches reorder based on search word similarity. Here, the user becomes interested in how the circadian rhythm might relate to Alzheimer's disease.
B. The webapp returns metapaths between Alzheimer's disease and the circadian rhythm pathway. The user unchecks "precomputed only" to compute results for all metapaths with length ≤ 3, not just those that surpass the database

inclusion threshold. The user sorts by adjusted *p*-value and selects 7 of the top 10 metapaths.
C. Paths for the selected metapaths are ordered by their path score. The user selects 8 paths (1 from a subsequent page of results) to show in the graph visualization and highlights a single path involving *ARNT2* for emphasis.
D. A subgraph displays the previously selected paths. The user improves on the automated layout by repositioning nodes. Clicking an edge displays its properties, informing the user that association between Creutzfeldt-Jakob disease and *NPAS2* was detected by GWAS.

Hetmatpy Package

We created the hetmatpy Python package, available on GitHub and PyPI under the permissive BSD-2-Clause Plus Patent License. This package provides a matrix-based utilities for hetnets.

TODO: improve flow and cohesion between software methods and results.

DWPC null distribution

To assess connectivity between a source and target node, we use the DWPC (degree-weighted path count) metric. The DWPC is similar to path count (number of paths between the source and target node along a given metapath), except that it downweights paths through high degree nodes. Rather than using the raw DWPC for a source-metapath-target combination, we transform the DWPC across all source-target node pairs for a metapath to yield a distribution that is more compact and amenable to modeling [35].

Previously, we had no technique for detecting whether a DWPC value was exceptional. One possibility is to evaluate the DWPCs for all pairs of nodes and select the top scores (e.g. the top 5% of DWPCs). Another possibility is to pick a transformed DWPC score as a cutoff. The shortcomings of these methods are twofold. First, neither the percentile nor absolute value of a DWPC has inherent meaning. To select transformed DWPCs greater than 6, or alternatively the top 1% of DWPCs, is arbitrary. Second, comparing DWPCs between node pairs fails to account for the situation where high-degree node pairs are likely to score higher, solely on account of their degree (TODO: figure).

To address these shortcomings, we developed a method to compute the right-tail *p*-value of a DWPC. *p*-values have a broadly understood interpretation — in our case, the probability that a DWPC equal to or greater than the observed DWPC could occur under a null model. By tailoring the null distribution for a DWPC to the degree of its source and target node, we account for degree effects when determining the significance of a DWPC.

Enriched metapaths

TODO: write this section

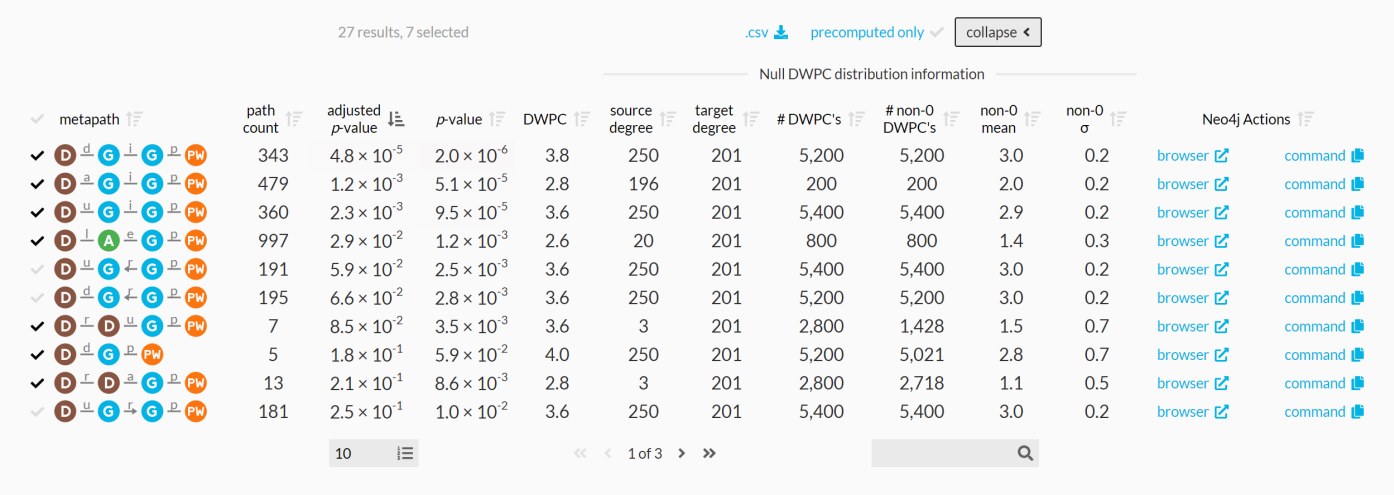

Figure 3: Expanded metapath details from the connectivity search webapp. This is the expanded view of the metapath table in 2B.

Figure 3 shows the information used to compute *p*-value for enriched metapaths. The table includes the following columns:

- **path count**: The number of paths between the source and target node of the specified metapath
- **adjusted *p*-value**: A measure of the significance of the DWPC that indicates whether more paths were observed than expected due to random chance. Compares the DWPC to a null distribution of DWPCs generated from degree-preserving permuted networks. Bonferroni-adjusted for the number of metapaths with the same source metanode, target metanode, and length.
- ***p*-value**: A measure of the significance of the DWPC that indicates whether more paths were observed than expected due to random chance. Compares the DWPC to a null distribution of DWPCs generated from degree-preserving permuted networks. Not adjusted for multiple comparisons (i.e. when multiple metapaths are assessed for significant connectivity between the source and target node).
- **DWPC**: Degree-Weighted Path Count — Measures the extent of connectivity between the source and target node for the given metapath. Like the path count, but with less weight given to paths along high-degree nodes.
- **source degree**: The number of edges from the source node that are of the same type as the initial metaedge of the metapath.
- **target degree**: The number of edges from the target node that are of the same type as the final metaedge of the metapath.
- **# DWPCs**: The number of DWPCs calculated on permuted networks used to generate a null distribution for the DWPC from the real network. Permuted DWPCs are aggregated for all permuted node pairs with the same degrees as the source and target node.
- **# non-0 DWPCs**: The number of permuted DWPCs from '# of DWPCs' column that were nonzero. Nonzero DWPCs indicate at least one path between the source and target node existed in the permuted network.
- **non-0 mean**: The mean of nonzero permuted DWPCs. Used to generate the gamma-hurdle model of the null DWPC distribution.
- **non-0 σ**: The standard deviation of nonzero permuted DWPCs. Used to generate the gamma-hurdle model of the null DWPC distribution.
- **Neo4j Actions**: A Cypher query that users can run in the Neo4j browser to show paths with the largest DWPCs for the metapath.

Enriched paths

TODO: write this section

The paths webapp panel includes the following information (Figure 2C):

- **path**: The sequence of edges in the network connecting the source node to the target node. Duplicate nodes are not permitted in paths.
- **path score**: A metric of how meaningful the path is in describing the connectivity between the source and target node. The score combines the magnitude of the metapath's p-value with the percent of the DWPC contributed by the path.
- **% of DWPC** The contribution of the path to the DWPC for its metapath. This metric compares the importance of all paths of the same metapath from the source node to the target node.

TODO: discuss path score and ranking by path score.

Comparison to Rephetio

TODO: write this section.

Use cases

TODO: determine what further use cases and examples we'd like to explore.

Detecting Mechanisms of Action for Indications

TODO: Assess ability to predict paths in https://github.com/SuLab/DrugMechDB

Discussion

In this study we introduce a search engine for hetnet connectivity between two nodes that is able to return results in realtime. An interactive webapp helps users explore node connectivity by ranking metapaths and paths, while visualizing multiple paths in a subgraph.

Several methodological contributions made this possible. We developed optimized algorithms for computing DWPCs using matrix multiplication. In addition, we created a method for estimating a *p*-value for a DWPC, using null DWPCs computed on permuted hetnets. We implemented these advances in the open source hetmatpy Python package and HetMat data structure to provide highly-optimized computational infrastructure for representing and reasoning on hetnets using matrices.

This work lays the foundation for exciting future directions. Here, we computed all DWPCs for Hetionet metapaths with length ≤ 3. Our search engine will therefore overlook important connectivity from longer metapaths. However, it is infeasible to compute DWPCs for all longer metapaths. One solution would be to only extend metapaths that are detected as informative. For example, if a *CbGpPWpG* metapath is deemed informative, then it could be extended with additional metaedges like *CbGpPWpGaD*. One unsupervised approach would be to use the distribution of DWPC *p*-values for a metapath to detect whether the paths still convey sufficient information, for example by requiring an enrichment of small *p*-values. Were this to method to fail, supervised alternatives could be explored, such as the ability for DWPCs from a longer metapath to predict that of a shorter metapath or metaedge, with care taken to prevent label leakage. One final approach could learn from user interest and compute longer metapaths only when requested.

In this work, we focus on queries where the input is a node pair. Equally interesting would be queries where the input is a set of nodes of the same type, optionally with weights. The search would compute DWPCs for paths originating on the query nodes. The simpler formulation would compute DWPCs for metapaths separately and compare to null distributions from permuted hetnets. A more advanced formulation would combine scores accross metapaths such that every node in the hetnet would receive a single score capturing its connectivity to the query set. This approach would have similar utility to gene set enrichment analysis (GSEA) in that the user could provide a set of genes as input and receive a ranked list of nodes that characterize the function of the query genes. However, it would excel in its versatility by returning results of any node type without requiring pre-defined gene sets to match against. Some users might be intested in node set transformations where scores for one node type are converted to another node type. This approach could take scores for human genes and convert them to side effects, diseases, pathways, etcetera.

Our work is not without limitation. The final product relies on multiple databases and cached computations that specific to Hetionet v1.0. Despite sriving for a modular architecture, generating an equivalent search webapp for a different hetnet would be challenging due to the many components involved. Furthermore, we would benefit from greater real-world evaluation of the connectivity search results to help identify situations where the method underperforms. Despite these challenges, our study demonstrates one of the first public search engines for node connectivity on a biomedical knowledge graph, while contributing methods and software that we hope will inspire future work.

Methods

The HetMat awakens

At the core of the hetmatpy package is the HetMat data structure for storing and accessing the network. HetMats are stored on disk as a directory, which by convention uses a `.hetmat` extension. A HetMat directory stores a single heterogeneous network, whose data resides in the following files.

1. A `metagraph.json` file stores the schema, defining which types of nodes and edges comprise the hetnet. This format is defined by the hetnetpy Python package. Hetnetpy was originally developed with the name hetio during prior studies [4,36], but we renamed it to het**net**py for better disambiguation from het**mat**py.
2. A `nodes` directory containing one file per node type (metanode) that defines each node. Currently, `.tsv` files where each row represents a node are supported.
3. An `edges` directory containing one file per edge type (metadata) that encodes the adjacency matrix. The matrix can be serialized using either the Numpy dense format (`.npy`) or SciPy sparse format (`.sparse.npz`).

For node and edge files, compression is supported as detected from `.gz` , `.bz2` , `.zip` , and `.xz` extensions. This structure of storing a hetnet supports selectively reading nodes and edges into memory. For example, a certain computation may only require access to a subset of the node and edge types. By only loading the required node and edge types, we reduce memory usage and read times.

Additional subdirectories, such as `path-counts` and `permutations` , store data generated from the HetMat. By using consistent paths for generated data, we avoid recomputing data that already exists on disk. A HetMat directory can be zipped for archiving and transfer. Users can selectively include generated data in archives. Since the primary application of HetMats is to generate computationally demanding measurements on hetnets, the ability to share HetMats with precomputed data is paramount.

The `HetMat` class implements the above logic. A `hetmat_from_graph` function creates a HetMat object and directory on disk from the pre-existing `hetnetpy.hetnet.Graph` format.

We converted Hetionet v1.0 to HetMat format and uploaded the `hetionet-v1.0.hetmat.zip` archive to the Hetionet data repository.

DWPC matrix multiplication algorithms

Prior to this study, we used two implementations for computing DWPCs. The first is a pure Python implementation available in the `hetnetpy.pathtools.DWPC` function [36]. The second uses a Cypher query, prepared by `hetnetpy.neo4j.construct_dwpc_query` , that is executed by the Neo4j database [4,37]. Both of these implementations require traversing all paths between the source and target node. Hence, they are computationally cumbersome despite optimizations [38].

Since our methods only require degree-weighted counts, not fully enumerated paths, adjacency matrix multiplication presents an alternative approach. Multiplication alone, however, counts walks rather than paths, meaning paths traversing a single node multiple times are counted. When computing network-based features to quantify the relationship between a source and target node, we would like to exclude traversing duplicate nodes (i.e. paths, not trails nor walks) [39]. To benefit from

the speed advantages of only counting paths, we developed a suite of algorithms to compute true path counts and DWPCs using matrix multiplication.

Our implementation begins by categorizing a metapath according to the pattern of its repeated metanodes, allowing DWPC computation using a specialized order of operations. For example, the metapath *DrDtCrC* is categorized as a set of disjoint repeats, while *DtCtDpC* is categorized as repeats of the form BABA. Many complex repeat patterns can be represented piecewise as simpler patterns, allowing us to compute DWPC for most metapaths up to length 5 and many of length 6 and beyond without enumerating individual paths. For example, disjoint groups of repeats like *DrDtCrC* can be computed as the matrix product of DWPC matrices for *DrD* and *CrC*. Randomly-inserted non-repeated metanodes (e.g. *G* in *DrDaGaDrD*) require no special treatment, and are included in DWPC with a simple matrix multiplication.

After metapath categorization, we segment metapaths according to their repeat pattern, following our order of operations. By segmenting and computing recursively, we can evalute DWPC efficiently on highly complex metapaths, using simple patterns as building-blocks for higher-level patterns. Finally, our specialized DWPC functions are applied to individual segments, the results are combined, and final corrections are made to ensure no repeated nodes are counted. The recursive, segmented approach we developed allowed us additionally to implement a caching strategy that improved speed by avoiding duplicate DWPC computations. In summary, the functionality we developed resulted in greater than a 175-fold reduction in compute time, allowing us to compute millions of DWPC values across Hetionet [40].

Details of matrix DWPC implementation

DWPC computation requires us to remove all duplicate nodes from paths. We used three repeat patterns as the building blocks for DWPC computation: short repeats (AAA), nested repeats (BAAB), and overlapping repeats (BABA). Let $\mathrm{D}(XwXyZ)$ denote the DWPC matrix for metapath *XwXyZ*. Under this notation, $\mathrm{D}(XyZ)$ is the degree-weighted (bi)adjacency matrix for metaedge *XyZ*. Additionally, let $\mathrm{diag}(A)$ represent a diagonal matrix whose entries are the diagonal elements of A.

For the case of short (< 4) repeats for a single metanode, *XaXbX* (e.g. *GiGdG*), we simply subtract the main diagonal.

$$\mathrm{D}(XaXbX) = \mathrm{D}(XaX)\mathrm{D}(XbX) - \mathrm{diag}(\mathrm{D}(XaX)\mathrm{D}(XbX))$$

Nested repeats *XaYbYcX* (e.g. *CtDrDtC*), are treated recursively, with both inner (YY) and outer (XX) repeats treated as separate short repeats.

$$\mathrm{D}(XaYbYcX) = \mathrm{D}(XaY)\mathrm{D}(YbY)\mathrm{D}(YcX) - \mathrm{diag}(\mathrm{D}(XaY)(\mathrm{D}(YbY)\mathrm{D}(YcX))$$

Overlapping repeats *XaYbXcY* (e.g. *CtDtCtD*) require several corrections ($\odot$ denotes the Hadamard product).

$$\begin{aligned}\mathrm{D}(XaYbXcY) = {} & \mathrm{D}(XaY)\,\mathrm{D}(YbX)\,\mathrm{D}(XcY) \\ & - \mathrm{diag}(\mathrm{D}(XaY)\,\mathrm{D}(YbX))\,\mathrm{D}(XcY) \\ & - \mathrm{D}(XaY)\,\mathrm{diag}(\mathrm{D}(YbX)\,\mathrm{D}(XcY)) \\ & + \mathrm{D}(XaY)\,\odot \mathrm{D}(YbX)^T\,\odot \mathrm{D}(XcY)\end{aligned}$$

Most paths of length six—and many even longer paths—can be represented hierarchically using these patterns. For example, a long metapath pattern of the form CBABACXYZ can be segmented as

(C(BABA)C)XYZ using patterns for short and overlapping repeats and can be computed using the tools we developed. In addition to these matrix routines—which advantageously count rather than enumerate paths—we implemented a general matrix method for any metapath type. The general method is important for patterns such as long (≥ 4) repeats, or complex repeat patterns (e.g. of the form ABCABC), but it requires path enumeration and is therefore slower. As an alternative approach for complex paths, we developed an approximate DWPC method that corrects repeats in disjoint simple patterns but only corrects the first repeat in complex patterns (e.g. ≥ length four repeat). Mayers et al. developed an alternative approximation, which subtracts the main diagonal at every occurrence of the first repeated metanode [41]. All our matrix methods were validated against existing implementations involving explicit path enumeration to ensure consistent results.

Permuted hetnets

In order to generate a null distribution for a DWPC, we rely on DWPCs computed from permuted hetnets. We derive permuted hetnets from the unpermuted network using the XSwap algorithm [42]. XSwap randomizes edges while preserving node degree. Therefore, it's ideal for generating null distributions that retain general degree effects, but destroy the actual meaning of edges. We adapt XSwap to hetnets by applying it separately to each metaedge [4,43,44].

Project Rephetio created 5 permuted hetnets [4,43], which were used to generate a null distribution of classifier performance for each metapath-based feature. Here, we aim to create a null distribution for individual DWPCs, which requires vastly more permuted values to estimate with accuracy. Therefore, we generated 200 permuted hetnets (archive). More recently, we also developed the `xswap` Python package, whose optimized C/C++ implementation will enable future research to generate even larger sets of permuted networks [44].

Degree-grouping of node pairs

For each of the 200 permuted networks and each of the 2,205 metapaths, we computed the entire DWPC matrix (i.e. all source nodes × target nodes). Therefore, for each actual DWPC value, we computed 200 permuted DWPC values. Because permutation preserves only node degree, DWPC values among nodes with the same source and target degrees are equivalent to additional permutations. We greatly increased the effective number of permutations by grouping DWPC values according to node degree, affording us a superior estimation of the DWPC null distribution.

We have applied this *degree-grouping* approach previously when calculating the prior probability of edge existence based on the source and target node degrees [44,45]. But here, we apply *degree-grouping* to null DWPCs. The result is that the null distribution for a DWPC is based not only on permuted DWPCs for the corresponding source–metapath–target combination, but instead on all permuted DWPCs for the source-degree–metapath–target-degree combination.

The "# DWPCs" column in Figure 3 illustrates how degree-grouping inflates the sample size of null DWPCs. The *p*-value for the *DaGiGpPW* metapath relies on the minimum number of null DWPCs (200), since no other disease besides Alzheimer's had 196 *associates* edges (source degree) and no other pathway besides circadian rhythm had 201 *participates* edges (target degree). However, for other metapaths with over 5,000 null DWPCs, degree-grouping increased the size of the null distribution by a factor of 25. In general, source–target node pairs with lower degrees receive the largest sample size multiplier from degree-grouping. This is convenient since low degree nodes also tend to produce the highest proportion of zero DWPCs, by virtue of low connectivity. Consequently, degree-grouping excels where it is needed most.

One final benefit of degree-grouping is that reduces the disk space required to store null DWPC summary statistics. For example, with 20,945 genes in Hetionet v1.0, there exists 438,693,025 gene pairs. Gene nodes have 302 distinct degrees for *interacts* edges, resulting in 91,204 degree pairs. This equates to an 4,810-fold reduction in the number of summary statistics that need to be stored to represent the null DWPC distribution for a metapath starting and ending with a *Gene–interacts–Gene* metaedge.

We store the following null DWPC summary statistics for each metapath–source-degree–target-degree combination: total number of null DWPCs, total number of nonzero null DWPCs, sum of null DWPCs, sum of squared null DWPCs, and number of permuted hetnets. These values are sufficient to estimate the *p*-value for a DWPC, by either fitting a gamma-hurdle null distribution or generating an empiric *p*-value. Furthermore, these statistics are additive across permuted hetnets. Their values are always a running total and can be updated incrementally as statistics from each additional permuted hetnet become available.

Gamma-hurdle distribution

We are interested in identifying source and target nodes whose connectivity exceeds what typically arises at random. To identify such especially-connected nodes, we compare DWPC values to the distribution of permuted network DWPC values for the same source and target nodes. While a single DWPC value is not actually a test statistic, we use a framework akin to classical hypothesis testing to identify outliers.

Two observations led us to the quasi significance testing framework we developed. First, a sizable fraction of permuted DWPC values are often zero, indicating that the source and target nodes are not connected along the metapath in the permuted network. Second, we observed that non-zero DWPC values for any given source and target nodes are reasonably approximated as following a gamma distribution. Motivated by these observations, we parametrized permuted DWPC values using a zero-inflated gamma distribution, which we termed the gamma-hurdle distribution. We fit a gamma-hurdle distribution to each combination of source node, target node, and metapath. Finally, we estimate the probability of observing a permuted DWPC value greater than DWPC computed in the unpermuted network, akin to a one-tailed p-value. These quasi significance scores ('*p*-values') allow us to identify outlier node pairs at the metapath level (see examples in Figure 4).

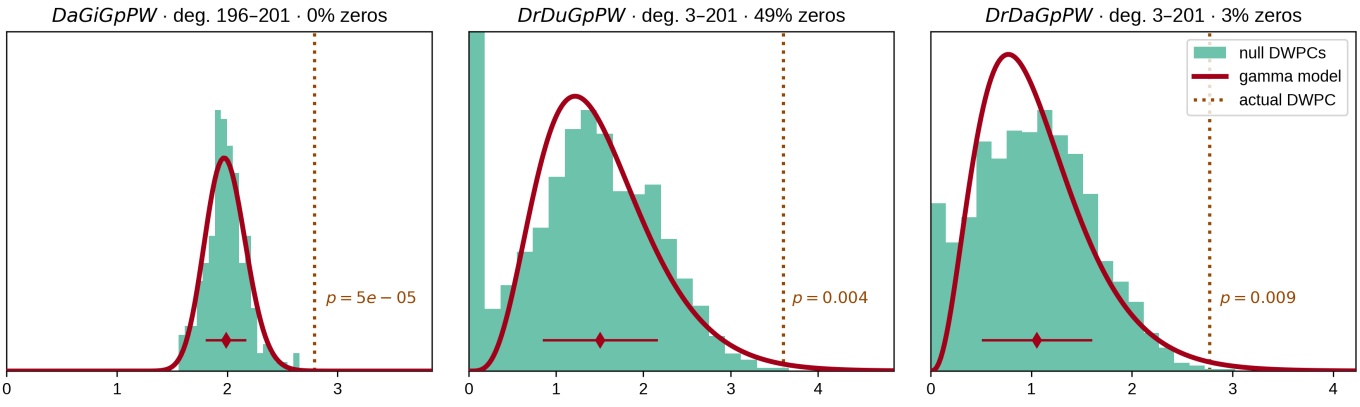

Figure 4: From null distribution to *p*-value for DWPCs. Null DWPC distributions are shown for 3 metapaths between Alzheimer's disease and the circadian rhythm pathway, selected from Figure 3. For each metapath, null DWPCs are computed on 200 permuted hetnets and grouped according to source–target degree. Histograms show the null DWPCs for the degree group corresponding to Alzheimer's disease and the circadian rhythm pathway (as noted in the plot titles by deg.) The proportion of null DWPCs that were zero is calculated, forming the "hurdle" of the null distribution model. The nonzero null DWPCs are modeled using a gamma distribution, which can be fit solely from a sample mean and standard deviation. The mean of nonzero null DWPCs is denoted with a diamond, with the standard deviation plotted twice as a line in either direction. Actual DWPCs are compared to the gamma-hurdle null distribution to yield a *p*-value.

Details of the gamma-hurdle distribution

Let X be a gamma-hurdle random variable with parameters λ, α, and β.

$$X \sim \Gamma_H(\lambda, \alpha, \beta)$$

The probability of a draw from the distribution is

$$P(X = 0) = 1 - \lambda$$
$$P(X \in A; A \subseteq (0, \infty)) = \frac{\lambda \beta^\alpha}{\Gamma(\alpha)} \int_{x \in A} \left(x^{\alpha - 1} e^{-\beta x} \right)$$

We estimate all three parameters using the method of moments (using Bessel's correction to estimate the second moment). As a validation of our method, we compared our method of moments parameter estimates to approximate maximum likelihood estimates (gamma distribution parameters do not have closed-form maximum likelihood estimates) and found excellent concordance between the methods. Let N be the number of permuted DWPC values, and n the number of nonzero values.

$$\hat{\lambda} = \frac{n}{N}$$
$$\hat{\alpha} = \frac{(n-1)\sum x_i}{n\sum(x_i^2) - (\sum x_i)^2}$$
$$\hat{\beta} = \frac{n-1}{n} \frac{(\sum x_i)^2}{n\sum(x_i)^2 - (\sum x_i)^2}$$

Finally, we compute a p-value for each DWPC value, t.

$$p = P(X \geq t) = \frac{\beta^\alpha}{\Gamma(\alpha)} \int_t^\infty x^{\alpha - 1} \exp(-\beta x) dx$$

Empirical DWPC p-values

We calculate an empirical p-value for special cases where the gamma-hurdle model cannot be applied. These cases include when the observed DWPC is zero or when the null DWPC distribution is all zeroes or has only a single distinct nonzero value. The empirical p-value ($p_{empiric}$) equals the proportion of null DPWCs ≥ the observed DWPC.

Since we don't store all null DWPC values, we apply the following criteria to calculate $p_{empiric}$ from summary statistics:

1. When the observed DWPC = 0 (no paths of the specified metapath existed between the source and target node), $p_{empiric} = 1$.
2. When all null DWPCs are zero but the observed DWPC is positive, $p_{empiric} = 0$.
3. When all nonzero null DWPCs have the same positive value (standard deviation = 0), $p_{empiric} = 0$ if the observed DWPC > the null DWPC else $p_{empiric}$ = proportion of nonzero null DWPCs.

DWPC and null distribution computation

We decided to compute DWPCs and their significance for all source–target node pairs for metapaths with length ≤ 3. On Hetionet v1.0, there are 24 metapaths of length 1, 242 metapaths of length 2, and 1,939 metapaths of length 3. The decision to stop at length 3 was one of practicality, as length 4 would have added 17,511 metapaths.

For each of the 2,205 metapaths, we computed the complete path count matrix and DWPC matrix (notebook). In total, we computed 137,786,767,964 path counts (and the same number of DWPCs) on the unpermuted network, of which 11.6% were nonzero.

The DWPC has a single parameter, called the damping exponent (w), which controls how much paths through high-degree nodes are downweighted [36]. When $w = 0$, the DWPC is equivalent to the path count. Previously, we found $w = 0.4$ was optimal for predicting disease-associated genes [36]. Here, we use $w = 0.5$, since taking the square root of degrees has more intuitive appeal.

We selected data types for matrix values that would allow for high precision. We used 64-bit unsigned integers for path counts and 64-bit floating-point numbers for DWPCs. We considered using 16-bits or 32-bits per DWPC to reduce memory/storage size, but decided against it in case certain applications required greater precision.

We used SciPy sparse for path count and DWPC matrices with density < 0.7, serialized to disk with compression and a `.sparse.npz` extension. This format minimizes the space on disk and load time for the entire matrix, but does not offer read access to slices. We used Numpy 2D arrays for DWPC matrices with density ≥ 0.7, serialized to disk using Numpy's `.npy` format. We bundled the path count and DWPC matrix files into HetMat archives by metapath length and deposited the archives to Zenodo [46]. The archive for length 3 DWPCs was the largest at 131.7 GB.

We also generated null DWPC summary statistics for the 2,205 metapaths, which are also available by metapath length from Zenodo as HetMat archives consisting of `.tsv.gz` files [46]. Due to degree-grouping, null DWPCs summary statistic archives are much smaller than the DWPC archives. The archive for length 3 null DWPCs summary statistics was 733.1 MB. However, the compute required to generate null DWPCs is far greater, because there are multiple permuted hetnets (in our case 200). As a result, computing and saving all DWPCs took 6 hours, whereas computing and saving the null DWPC summary statistics took 361 hours.

Including null DWPCs and path counts, the Zenodo deposit totals 185.1 GB and contains the results of computing ~28 trillion DWPCs — 27,832,927,128,728 to be exact.

Adjusting DWPC *p*-values

When a user applies hetnet connecitivity search to identify enriched metapaths between two nodes, many metapaths are evaluated for significance. Due to multiple testing of many DWPCs, low *p*-values are likely to arise by chance. Therefore, we devised a multiple testing correction.

For each combination of source metanode, target metanode, and length, we counted the number of metapaths. For Disease…Pathway metapaths, there are 0 metapaths of length 1, 3 metapaths of length 2, and 24 metapaths of length 3. We calculated adjusted p-values by applying a Bonferroni correction based on the number of metapaths of the same length between the source and target metanode. Using Figure 3 as an example, the DdGpPW p-value of 5.9% was adjusted to 17.8% (multiplied by a factor of 3).

Bonferroni controls familywise error rate, which corresponds here to incorrectly finding that *any* metapath of a given length is enriched. As a result, our adjusted p-values are conservative. We would

prefer to adjust *p*-values for false discovery rate [47], but these methods often require access to all *p*-values at once (impractical here) and assume a uniform distribution of *p*-values when there is no signal (not the case here when most DWPCs are zero).

Prioritizing enriched metapaths for database storage

Storing DWPCs and their significance in the database (as part of the `PathCount` table in Figure 5) enables the connectivity search webapp to provide users with enriched metapaths between query nodes in realtime. However, storing ~15.9 billion rows (the total number of nonzero DWPCs) in the database's `PathCount` table would exceed a reasonable disk quota. An alternative would be to store all DWPCs in the database whose adjusted p-value exceeded a universal threshold (e.g. $p < 5\%$). But we estimated this would still be prohibitively expensive. Therefore, we devised a metapath-specific threshold. For metapaths with length 1, we stored all nonzero DWPCs, assuming users always want to be informed about direct edges between the query nodes, regardless of significance. For metapaths with length ≥ 2, we chose an adjusted *p*-value threshold of $5 \times (n_{source} \times n_{target})^{-0.3}$, where n_{source} and n_{target} are the node counts for the source and target metanodes (i.e. "Nodes" column in Table 1). Notice that metapaths with large number of possible source–target pairs (large DWPC matrices) are penalized. This decision is based on practicality, since otherwise the majority of the database quota would be consumed by a minority of metapaths between plentiful metanodes (e.g. Gene…Gene metapaths). Also we assume that users will search nodes at a similar rate by metanode (e.g. they're more likely to search for a specific disease than a specific gene). The constants in the threshold formula help scale it. The multiplier of 5 relaxes the threshold to saturate the available database capacity. The −0.3 exponent applies the large DWPC-matrix penalty.

Users can still evaluate DWPCs that are not stored in the database, using either the webapp or API. These are calculated on-the-fly, delegating DWPC computation to the Neo4j database. Unchecking "precomputed only" on the webapp shows all possible metapaths for two query nodes. For some node pairs, the on-the-fly computation is quick (less than a second). Othertimes, computing DWPCs for all metapaths might take more than a minute.

Backend Database & API

We created a backend application using Python's Django web framework. The source code is available in the `connectivity-search-backend` repository. The primary role of the backend is to manage a relational database and provide an API for requesting data.

We define the database schema using Django's object-relational mapping framework (Figure 5). We import the data into a PostgreSQL database. Populating the database for all 2,205 metapaths up to length 3 was a prolonged operation, taking over 3 days. The majority of the time is spent populating the `DegreeGroupedPermutation` (37,905,389 rows) and `PathCount` (174,986,768 rows) tables. To avoid redundancy, the database only stores a single orientation of a metapath. For example, if rows are stored for the *GpPWpGaD* metapath, they would not also be stored for the *DaGpPWpG* metapath. The backend is responsible for checking both orientations of a metapath in the database and reversing metapaths on-the-fly before returning results. The database is located at `search-db.het.io` with public read-only access available.

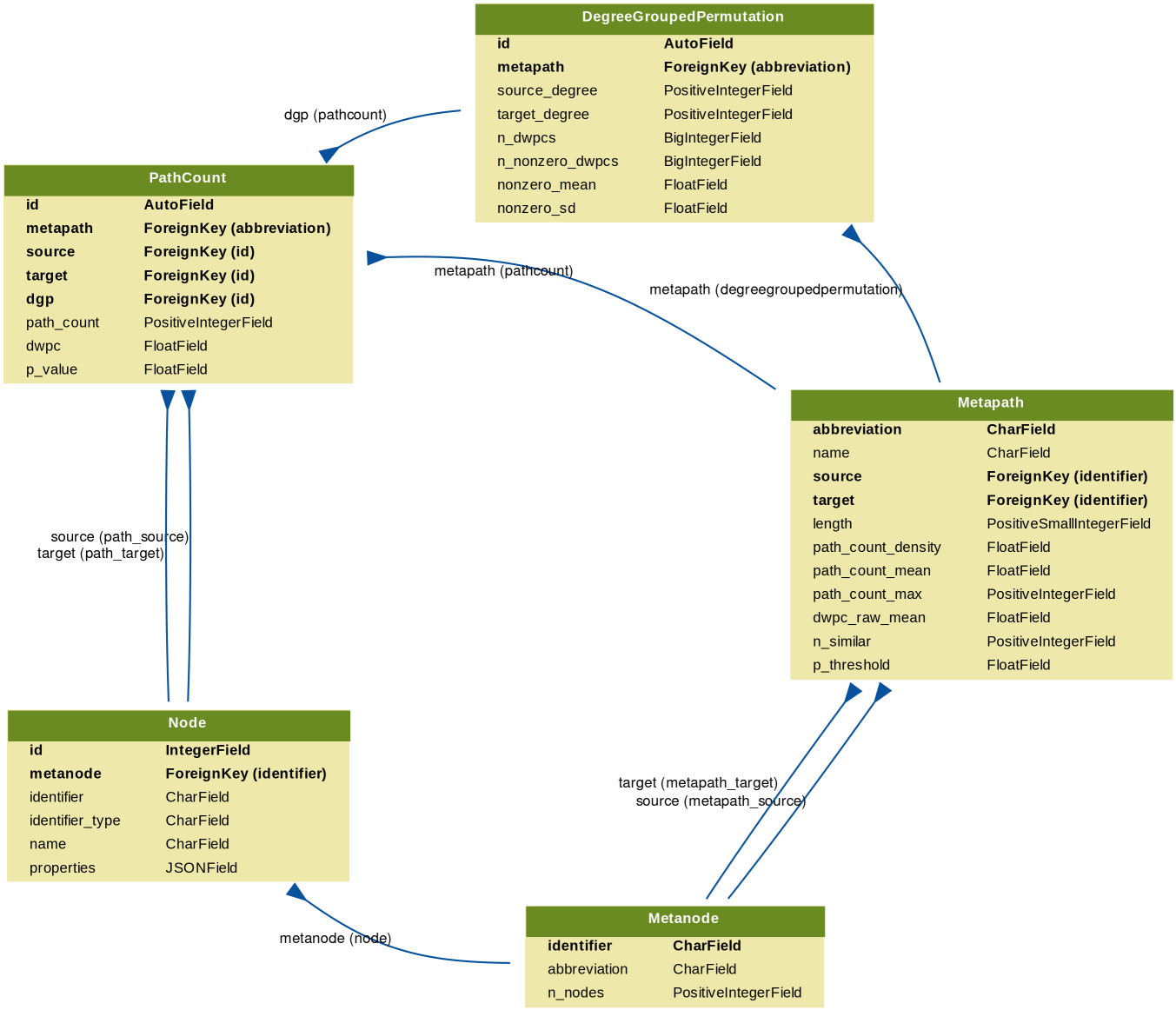

Figure 5: Schema for the connectivity search backend relational database models. Each Django model is represented as a table, whose rows list the model's field names and types. Each model corresponds to a database table. Arrows denote foreign key relationships. The arrow labels indicate the foreign key field name followed by reverse relation names generated by Django (in parentheses).

We host a public API instance at https://search-api.het.io. Version 1 of the API exposes several endpoints that are used by the connectivity search frontend including queries for node details (`/v1/node`), node lookup (`/v1/nodes`), metapath information (`/v1/metapaths`), and path information (`/v1/paths`). The endpoints return JSON payloads. Producing results for these queries relies on internal calls to the PostgreSQL relational database as well as the pre-existing Hetionet v1.0 Neo4j graph database. They were designed to power the hetnet connectivity search webapp, but are also available for general research use.

Webapp & Frontend

TODO: write this section. https://github.com/greenelab/connectivity-search-manuscript/issues/29

Realtime open science

This study was conducted entirely in the open via public GitHub repositories. We used GitHub Issues for discussion, leaving a rich online history of the scholarly process. Furthermore, most additions to

the analyses were performed by pull request, whereby a contributor proposes a set of changes. This provides an oppertunity for other contributors to review changes before they are officially accepted. For example, in greenelab/hetmech#156 @zietzm proposed a notebook to visualize parameters for null DWPC distributions. After @zietzm addressed @dhimmel's comments, the pull request was approved and merged into the project's main branch.

The manuscript for this study was written using Manubot, which allows authors to collaboratively write manuscripts on GitHub [48]. The Manubot-rendered manuscript is available at https://greenelab.github.io/connectivity-search-manuscript/. We encourage readers with feedback or questions to comment publicly via GitHub Issues.

Software & data availability

This study primarily involves the following repositories:

- greenelab/connectivity-search-manuscript: Source code for this manuscript. Best place for general comments or questions. CC BY 4.0 License.
- greenelab/hetmech: The initial project repository that contains research notebooks, dataset generation code, and exploratory data analyses. The hetmatpy package was first developed as part of this repository until its relocation in November 2018. BSD 3-Clause License.
- greenelab/connectivity-search-backend: Source code for the connectivity search database and API. BSD 3-Clause License.
- greenelab/connectivity-search-frontend: Source code for the connectivity search webapp. BSD 3-Clause License.
- hetio/hetmatpy: Python package for matrix storage and operations on hetnets. Released on PyPI. BSD 2-Clause Plus Patent License.
- hetio/hetnetpy Preexisiting python package for representing hetnets. Dependency of hetmatpy. Released on PyPI. Dual licensed under BSD 2-Clause Plus Patent License and CC0 1.0 (public domain dedication).
- hetio/hetionet. Preexisiting data repository for Hetionet, including the public Neo4j instance and HetMat archives. CC0 1.0 License.
- hetio/het.io. Preexisiting source code for the https://het.io/ website. CC BY 4.0 License.

The hetmech and hetionet repositories contain datasets related to this study. Large datasets were compressed and tracked with Git LFS (Large File Storage). GitHub LFS had a max file size of 2 GB. Datasets exceeding this size, along with other essential datasets, are available from Zenodo [46].

References

1. **Renaming 'heterogeneous networks' to a more concise and catchy term**
Daniel Himmelstein, Casey Greene, Sergio Baranzini
ThinkLab (2015-08-16) https://doi.org/f3mn4v
DOI: 10.15363/thinklab.d104

2. **Human Disease Ontology 2018 update: classification, content and workflow expansion**
Lynn M Schriml, Elvira Mitraka, James Munro, Becky Tauber, Mike Schor, Lance Nickle, Victor Felix, Linda Jeng, Cynthia Bearer, Richard Lichenstein, … Carol Greene
Nucleic Acids Research (2019-01-08) https://doi.org/ggx9wp
DOI: 10.1093/nar/gky1032 · PMID: 30407550 · PMCID: PMC6323977

3. **Unifying disease vocabularies**
Daniel Himmelstein, Tong Shu Li
ThinkLab (2015-03-30) https://doi.org/f3mqv5
DOI: 10.15363/thinklab.d44

4. **Systematic integration of biomedical knowledge prioritizes drugs for repurposing**
Daniel Scott Himmelstein, Antoine Lizee, Christine Hessler, Leo Brueggeman, Sabrina L Chen, Dexter Hadley, Ari Green, Pouya Khankhanian, Sergio E Baranzini
eLife (2017-09-22) https://doi.org/cdfk
DOI: 10.7554/elife.26726 · PMID: 28936969 · PMCID: PMC5640425

5. **Wikidata as a knowledge graph for the life sciences**
Andra Waagmeester, Gregory Stupp, Sebastian Burgstaller-Muehlbacher, Benjamin M Good, Malachi Griffith, Obi L Griffith, Kristina Hanspers, Henning Hermjakob, Toby S Hudson, Kevin Hybiske, … Andrew I Su
eLife (2020-03-17) https://doi.org/ggqqc6
DOI: 10.7554/elife.52614 · PMID: 32180547 · PMCID: PMC7077981

6. **SemMedDB: a PubMed-scale repository of biomedical semantic predications**
H Kilicoglu, D Shin, M Fiszman, G Rosemblat, TC Rindflesch
Bioinformatics (2012-10-08) https://doi.org/f4hp3x
DOI: 10.1093/bioinformatics/bts591 · PMID: 23044550 · PMCID: PMC3509487

7. **Constructing Biomedical Knowledge Graph Based on SemMedDB and Linked Open Data**
Qing Cong, Zhiyong Feng, Fang Li, Li Zhang, Guozheng Rao, Cui Tao
2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) (2018-12)
https://doi.org/ggzb26
DOI: 10.1109/bibm.2018.8621568

8. **Time-resolved evaluation of compound repositioning predictions on a text-mined knowledge network**
Michael Mayers, Tong Shu Li, Núria Queralt-Rosinach, Andrew I Su
BMC Bioinformatics (2019-12-11) https://doi.org/ggpcsr
DOI: 10.1186/s12859-019-3297-0 · PMID: 31829175 · PMCID: PMC6907279

9. **Announcing PharmacotherapyDB: the Open Catalog of Drug Therapies for Disease**
Daniel Himmelstein
ThinkLab (2016-03-15) https://doi.org/f3mqtv
DOI: 10.15363/thinklab.d182

10. **Our hetnet edge prediction methodology: the modeling framework for Project Rephetio**

Daniel Himmelstein
ThinkLab (2016-05-04) https://doi.org/f3qbmj
DOI: 10.15363/thinklab.d210

11. **The link-prediction problem for social networks**
David Liben-Nowell, Jon Kleinberg
Journal of the American Society for Information Science and Technology (2007-05)
https://doi.org/c56765
DOI: 10.1002/asi.20591

12. **Link prediction in complex networks: A survey**
Linyuan Lü, Tao Zhou
Physica A: Statistical Mechanics and its Applications (2011-03) https://doi.org/dbs8g8
DOI: 10.1016/j.physa.2010.11.027

13. **Heterogeneous network embedding for identifying symptom candidate genes**
Kuo Yang, Ning Wang, Guangming Liu, Ruyu Wang, Jian Yu, Runshun Zhang, Jianxin Chen, Xuezhong Zhou
Journal of the American Medical Informatics Association (2018-11) https://doi.org/gfg6nr
DOI: 10.1093/jamia/ocy117 · PMID: 30357378 · PMCID: PMC7646926

14. **Large-scale structural and textual similarity-based mining of knowledge graph to predict drug–drug interactions**
Ibrahim Abdelaziz, Achille Fokoue, Oktie Hassanzadeh, Ping Zhang, Mohammad Sadoghi
Journal of Web Semantics (2017-05) https://doi.org/gcrwk3
DOI: 10.1016/j.websem.2017.06.002

15. **SMR: Medical Knowledge Graph Embedding for Safe Medicine Recommendation**
Fan Gong, Meng Wang, Haofen Wang, Sen Wang, Mengyue Liu
Big Data Research (2021-02) https://doi.org/gjqwnc
DOI: 10.1016/j.bdr.2020.100174

16. **PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings**
Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, Jens Lehmann
Journal of Machine Learning Research (2021) http://jmlr.org/papers/v22/20-825.html

17. **Understanding the Performance of Knowledge Graph Embeddings in Drug Discovery**
Stephen Bonner, Ian P Barrett, Cheng Ye, Rowan Swiers, Ola Engkvist, Charles Tapley Hoyt, William L Hamilton
arXiv (2021-06-08) https://arxiv.org/abs/2105.10488

18. **node2vec**
Aditya Grover, Jure Leskovec
Association for Computing Machinery (ACM) (2016-08-13) https://doi.org/gftdzj
DOI: 10.1145/2939672.2939754 · PMID: 27853626 · PMCID: PMC5108654

19. **metapath2vec: Scalable Representation Learning for Heterogeneous Networks**
Yuxiao Dong, Nitesh V Chawla, Ananthram Swami
KDD '17: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2017-08) https://doi.org/gfsqzn
DOI: 10.1145/3097983.3098036

20. **edge2vec: Representation learning using edge semantics for biomedical knowledge discovery**

Zheng Gao, Gang Fu, Chunping Ouyang, Satoshi Tsutsui, Xiaozhong Liu, Jeremy Yang, Christopher Gessner, Brian Foote, David Wild, Ying Ding, Qi Yu
BMC Bioinformatics (2019-06-10) https://doi.org/ggpcsq
DOI: 10.1186/s12859-019-2914-2 · PMID: 31238875 · PMCID: PMC6593489

21. **Preclinical validation of therapeutic targets predicted by tensor factorization on heterogeneous graphs**
Saee Paliwal, Alex de Giorgio, Daniel Neil, Jean-Baptiste Michel, Alix MB Lacoste
Scientific Reports (2020-10-26) https://doi.org/gg49
DOI: 10.1038/s41598-020-74922-z · PMID: 33106501 · PMCID: PMC7589557

22. **Data Fusion by Matrix Factorization**
Marinka Zitnik, Blaz Zupan
IEEE Transactions on Pattern Analysis and Machine Intelligence (2015-01-01) https://doi.org/f3mqwz
DOI: 10.1109/tpami.2014.2343973 · PMID: 26353207

23. **Translating embeddings for modeling multi-relational data**
Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, Oksana Yakhnenko
Proceedings of the 26th International Conference on Neural Information Processing Systems (2013-12) https://dl.acm.org/doi/10.5555/2999792.2999923

24. **Predicting gene-disease associations from the heterogeneous network using graph embedding**
Xiaochan Wang, Yuchong Gong, Jing Yi, Wen Zhang
Institute of Electrical and Electronics Engineers (IEEE) (2019-11) https://doi.org/ggmrpj
DOI: 10.1109/bibm47256.2019.8983134

25. **A Method to Learn Embedding of a Probabilistic Medical Knowledge Graph: Algorithm Development**
Linfeng Li, Peng Wang, Yao Wang, Shenghui Wang, Jun Yan, Jinpeng Jiang, Buzhou Tang, Chengliang Wang, Yuting Liu
JMIR Medical Informatics (2020-05-21) https://doi.org/ghqszs
DOI: 10.2196/17645 · PMID: 32436854 · PMCID: PMC7273238

26. **Semantic Disease Gene Embeddings (SmuDGE): phenotype-based disease gene prioritization without phenotypes**
Mona Alshahrani, Robert Hoehndorf
Bioinformatics (2018-09-01) https://doi.org/gd9k8n
DOI: 10.1093/bioinformatics/bty559 · PMID: 30423077 · PMCID: PMC6129260

27. **A network embedding model for pathogenic genes prediction by multi-path random walking on heterogeneous network**
Bo Xu, Yu Liu, Shuo Yu, Lei Wang, Jie Dong, Hongfei Lin, Zhihao Yang, Jian Wang, Feng Xia
BMC Medical Genomics (2019-12-23) https://doi.org/ggmrpq
DOI: 10.1186/s12920-019-0627-z · PMID: 31865919 · PMCID: PMC6927107

28. **Deep mining heterogeneous networks of biomedical linked data to predict novel drug–target associations**
Nansu Zong, Hyeoneui Kim, Victoria Ngo, Olivier Harismendy
Bioinformatics (2017-08-01) https://doi.org/gbqjgx
DOI: 10.1093/bioinformatics/btx160 · PMID: 28430977 · PMCID: PMC5860112

29. **Explaining and Suggesting Relatedness in Knowledge Graphs**
Giuseppe Pirrò

Lecture Notes in Computer Science (2015) https://doi.org/gg5nkd
DOI: 10.1007/978-3-319-25007-6_36

30. **FAIRY**
Azin Ghazimatin, Rishiraj Saha Roy, Gerhard Weikum
Association for Computing Machinery (ACM) (2019-01-30) https://doi.org/gf2wqj
DOI: 10.1145/3289600.3290990

31. **Using Knowledge Graphs to Explain Entity Co-occurrence in Twitter**
Yiwei Wang, Mark James Carman, Yuan-Fang Li
Association for Computing Machinery (ACM) (2017-11-06) https://doi.org/gg5nkf
DOI: 10.1145/3132847.3133161

32. **ESPRESSO**
Stephan Seufert, Klaus Berberich, Srikanta J Bedathur, Sarath Kumar Kondreddi, Patrick Ernst, Gerhard Weikum
Association for Computing Machinery (ACM) (2016-10-24) https://doi.org/gcpx7w
DOI: 10.1145/2983323.2983778

33. **MetaExp: Interactive Explanation and Exploration of Large Knowledge Graphs**
Freya Behrens, Fatemeh Aghaei, Emmanuel Müller, Martin Preusse, Nikola Müller, Michael Hunger, Sebastian Bischoff, Pius Ladenburger, Julius Rückin, Laurenz Seidel, … Davide Mottin
WWW '18: Companion Proceedings of the The Web Conference 2018 (2018-04)
https://doi.org/gg2c7w
DOI: 10.1145/3184558.3186978

34. **Discovering Meta-Paths in Large Heterogeneous Information Networks**
Changping Meng, Reynold Cheng, Silviu Maniu, Pierre Senellart, Wangda Zhang
Association for Computing Machinery (ACM) (2015-05-18) https://doi.org/gg2c7v
DOI: 10.1145/2736277.2741123

35. **Transforming DWPCs for hetnet edge prediction**
Daniel Himmelstein, Pouya Khankhanian, Antoine Lizee
ThinkLab (2016-04-01) https://doi.org/f3qbmd
DOI: 10.15363/thinklab.d193

36. **Heterogeneous Network Edge Prediction: A Data Integration Approach to Prioritize Disease-Associated Genes**
Daniel S Himmelstein, Sergio E Baranzini
PLOS Computational Biology (2015-07-09) https://doi.org/98q
DOI: 10.1371/journal.pcbi.1004259 · PMID: 26158728 · PMCID: PMC4497619

37. **Using the neo4j graph database for hetnets**
Daniel Himmelstein
ThinkLab (2015-10-02) https://doi.org/f3mqvk
DOI: 10.15363/thinklab.d112

38. **Estimating the complexity of hetnet traversal**
Daniel Himmelstein, Antoine Lizee
ThinkLab (2016-03-22) https://doi.org/gbr42x
DOI: 10.15363/thinklab.d187

39. **Path exclusion conditions**
Daniel Himmelstein
ThinkLab (2015-12-08) https://doi.org/gg2rw2
DOI: 10.15363/thinklab.d134

40. **Vagelos Report Summer 2017**
Michael Zietz
figshare (2017) https://doi.org/gbr3pf
DOI: 10.6084/m9.figshare.5346577

41. **GitHub - mmayers12/hetnet_ml: Software to quickly extract features from heterogeneous networks for machine learning.**
GitHub
https://github.com/mmayers12/hetnet_ml

42. **Randomization Techniques for Graphs**
Sami Hanhijärvi, Gemma C Garriga, Kai Puolamäki
Society for Industrial & Applied Mathematics (SIAM) (2009-04-30) https://doi.org/f3mn58
DOI: 10.1137/1.9781611972795.67

43. **Assessing the effectiveness of our hetnet permutations**
Daniel Himmelstein
ThinkLab (2016-02-25) https://doi.org/f3mqt5
DOI: 10.15363/thinklab.d178

44. **The probability of edge existence due to node degree: a baseline for network-based predictions**
Michael Zietz, Daniel S Himmelstein, Kyle Kloster, Christopher Williams, Michael W Nagle, Blair D Sullivan, Casey S Greene
Manubot (2020-03-05) https://greenelab.github.io/xswap-manuscript/

45. **Network Edge Prediction: Estimating the prior**
Antoine Lizee, Daniel Himmelstein
ThinkLab (2016-04-14) https://doi.org/f3qbmg
DOI: 10.15363/thinklab.d201

46. **Node connectivity measurements for Hetionet v1.0 metapaths**
Daniel Himmelstein, Michael Zietz, Kyle Kloster, Michael Nagle, Blair Sullivan, Casey Greene
Zenodo (2018-11-06) https://doi.org/cww7
DOI: 10.5281/zenodo.1435833

47. **A practical guide to methods controlling false discoveries in computational biology**
Keegan Korthauer, Patrick K Kimes, Claire Duvallet, Alejandro Reyes, Ayshwarya Subramanian, Mingxiang Teng, Chinmay Shukla, Eric J Alm, Stephanie C Hicks
Genome Biology (2019-06-04) https://doi.org/gf3ncd
DOI: 10.1186/s13059-019-1716-1 · PMID: 31164141 · PMCID: PMC6547503

48. **Open collaborative writing with Manubot**
Daniel S Himmelstein, Vincent Rubinetti, David R Slochower, Dongbo Hu, Venkat S Malladi, Casey S Greene, Anthony Gitter
PLOS Computational Biology (2019-06-24) https://doi.org/c7np
DOI: 10.1371/journal.pcbi.1007128 · PMID: 31233491 · PMCID: PMC6611653