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

This manuscript (permalink) was automatically generated from greenelab/connectivity-search-manuscript@2615563 on June 25, 2020.

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 · Funded by GBMF4552

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 know how two nodes are meaningfully connected. For example, users may want to know 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].

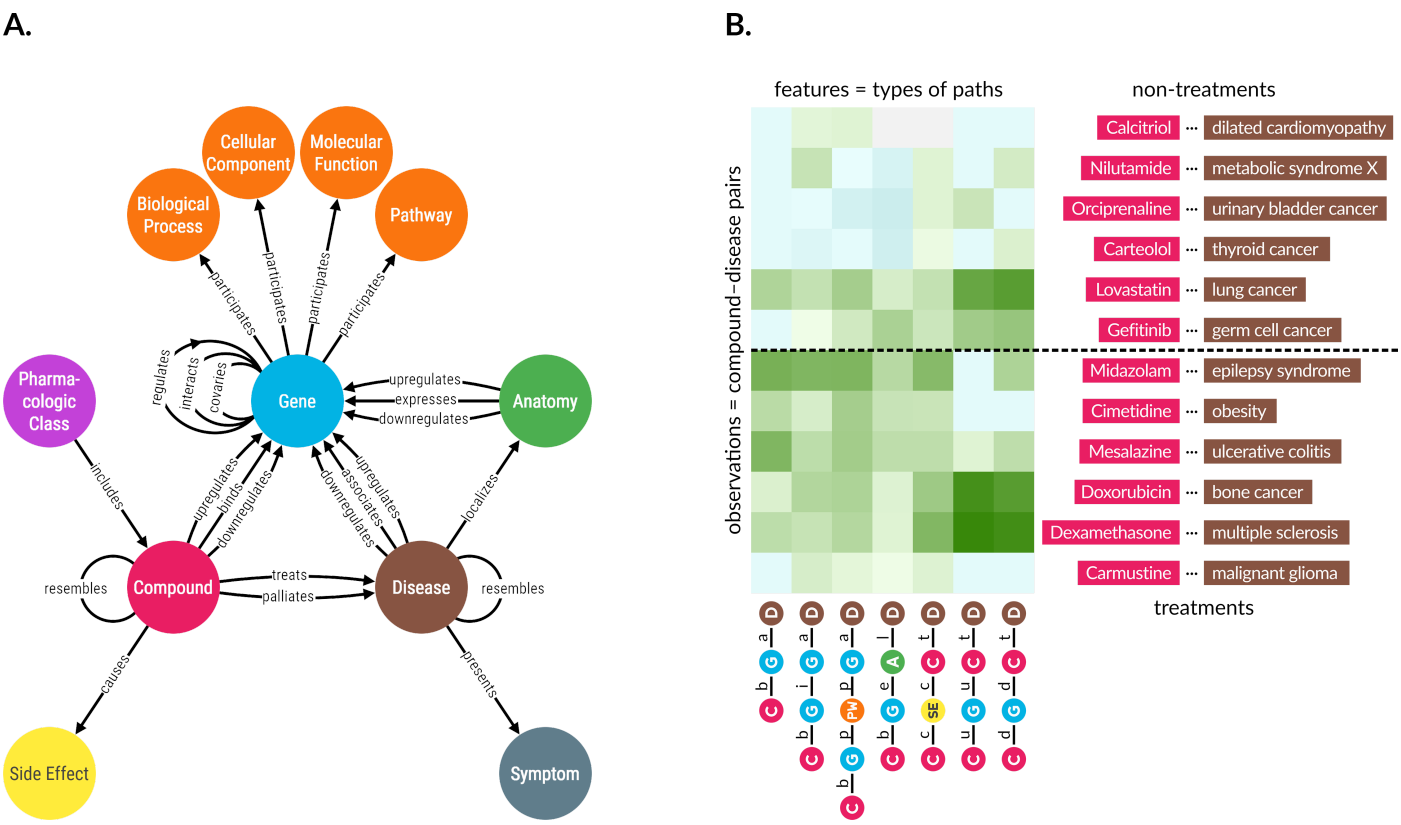

Figure 1: A. Hetionet v1.0 metagraph. The types of nodes and edges in Hetionet.
B. Supervised machine learning approach from Project Rephetio. This figure visualizes the feature matrix used by Project Rephetio to make supervised predictions. Each row represents a compound–disease pair. The top half of rows correspond to known treatments (i.e. positives), while the bottom half correspond to non-treatments (i.e. negatives, 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.

TODO: Other works

https://github.com/greenelab/hetmech/issues/56

Network embeddings edge2vec [11], metapath2vec [12], HINE [13].

14 training node pairs to important metapaths (Forward Stagewise Path Generation). MetaExp [15] user selects two sets of nodes. MetaExp detects metapaths and interacts with the user to progressively refine metapaths.

Unsupervised connectivity search

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.

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 [16].

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

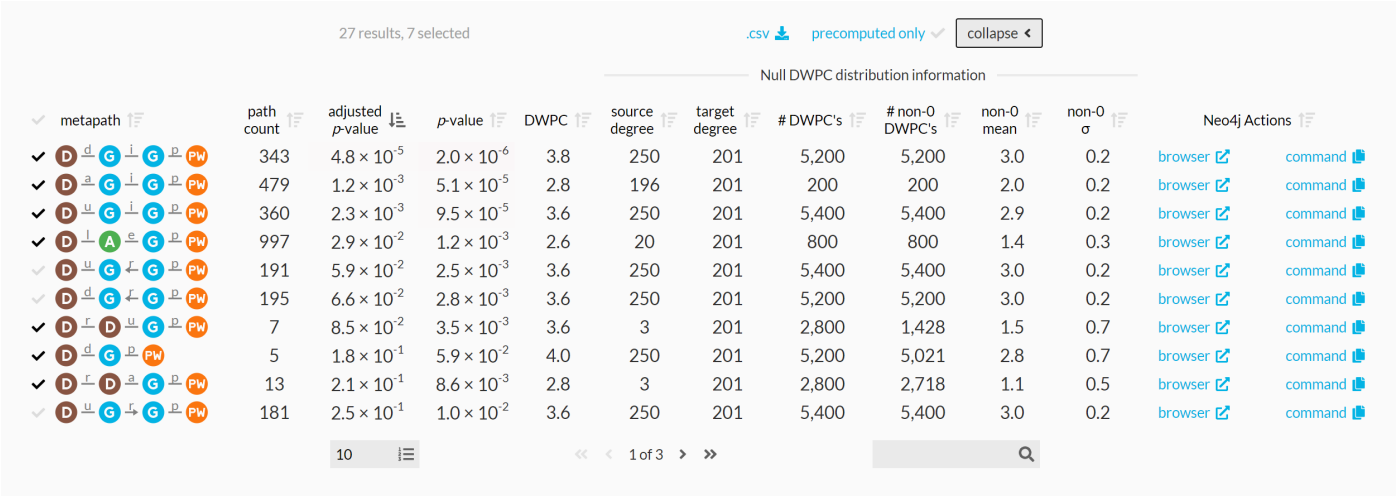

Figure 3: Expanded metapath details from the connectivity search webapp. This is the expanded view of the metapath table in 2B.

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

Comparison to Rephetio

Detecting Mechanisms of Action for Indications

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

Use cases

Discussion

STUB: Contributions of this work:

- search engine for hetnet connectivity between two nodes, realtime results
- interactive webapp and user interface for displaying metapaths, paths, and subgraphs.
- optimized methods for computing DWPCs using matrix multiplication
- method for estimating *p*-values for a DWPC, based on null DWPCs computed from permuted hetnets.
- the hetmatpy Python package and HetMat data structure that provide a highly-optimized computational infrastructure to make this possible.

STUB: Future work:

- node set transformations
- improved DWPC scaling
- longer metapaths
- auto-detection of informative metapaths

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 [4,17], 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 [17]. The second uses a Cypher query, prepared by `hetnetpy.neo4j.construct_dwpc_query` , that is executed by the Neo4j database [4,18]. Both of these implementations require traversing all paths between the source and target node. Hence, they are computationally cumbersome despite optimizations [19].

An alternative approach counts paths by multiplying adjacency matrices. However, this approach actually counts walks, since it includes sequences of edges that traverse a single node (i.e trail) or edge (i.e. walk) multiple times. 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) [20]. Therefore, we invented a suite of algorithms to compute true path counts and DWPCs using matrix multiplication.

TODO: Describe the suite of DWPC algorithms. From the categorize function, there are:

- no_repeats
- disjoint
- short_repeat
- long_repeat
- BAAB
- BABA
- repeat_around
- interior_complete_group
- disjoint_groups
- other

Include information on what lengths of metapath we have completely solved this for. Mention testing against existing methods.

Mention approximations. Also note independent approximation work at https://github.com/mmayers12/hetnet_ml

Discuss caching strategies, sequential versus recursive

Runtime comparison to show the speedup. Rephetio computed a portion of in XX time

From [21]:

> We reduced the time to compute DWPC over nearly 1200 metapaths from roughly four-and-a-half days to roughly one hour and thirty-seven minutes

175-fold, which underestimates since Rephetio did not compute the full DWPC matrix and benefited from concurrency.

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 [22]. 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 [4,23,24].

Project Rephetio created 5 permuted hetnets [4,23], 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 [24].

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 [24,25]. 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

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 [17]. When $w = 0$, the DWPC is equivalent to the path count. Previously, we found $w = 0.4$ was optimal for predicting disease-associated genes [17]. 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 [26]. 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 [26]. 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.

Prioritizing enriched metapaths for database storage

Storing ~15.9 billion rows in the database's `PathCount` table would exceed a reasonable disk quota.

Rest API & backend

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 manage a relational database and provide an API for requesting data.

We define the database schema using Django's object-relational mapping framework (Figure 4). 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.

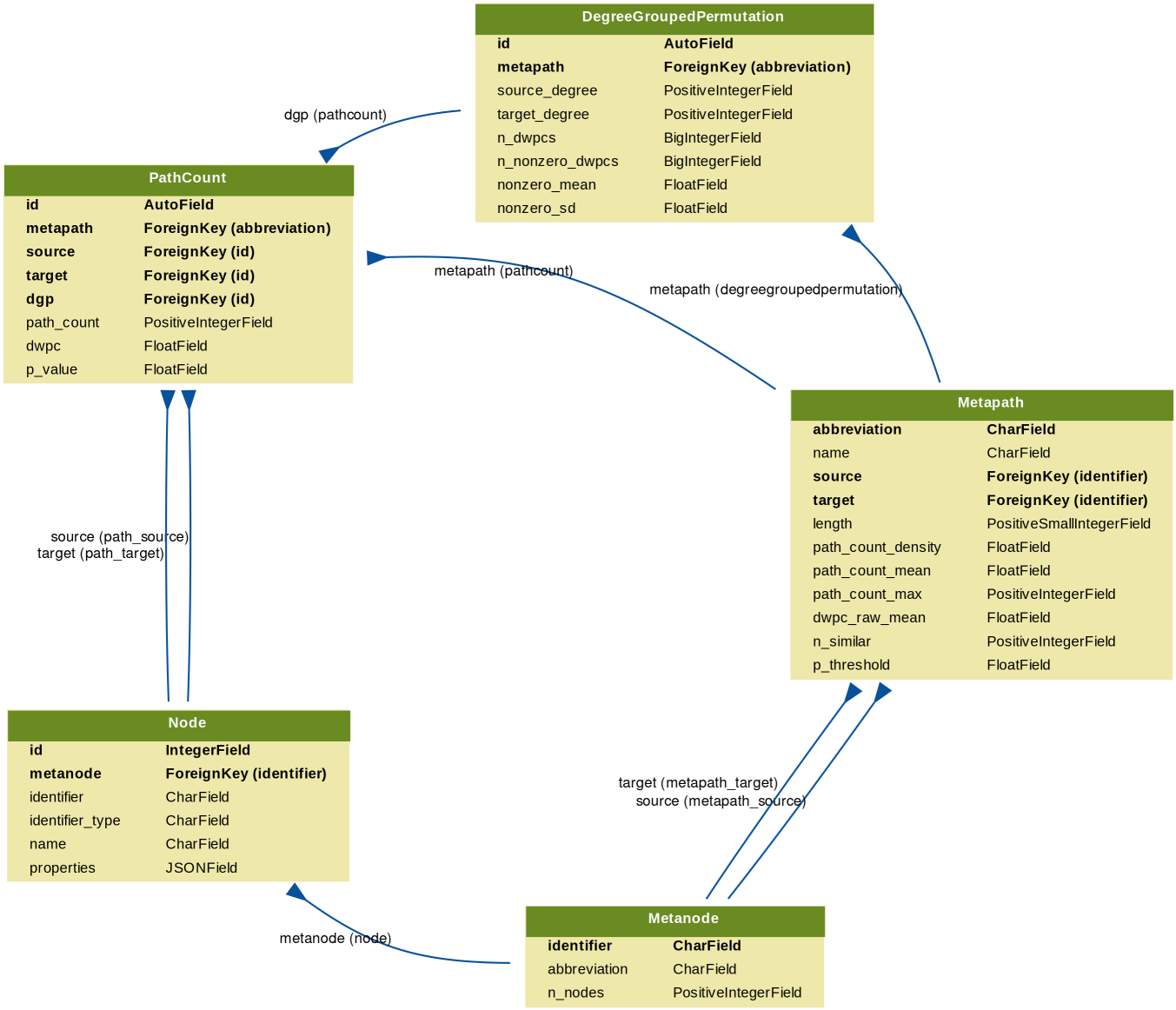

Figure 4: Schema for the connectivity search backend relational database models. Each Django model is represented as a table, whose rows list the model's field names and types. Each model corresponds to a database table. Arrows denote foreign key relationships. The arrow labels indicate the foreign key 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

Realtime open science

Software & data availability

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, T. C. 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. **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

12. **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

13. **HINE: Heterogeneous Information Network Embedding**
Yuxin Chen, Chenguang Wang
Lecture Notes in Computer Science (2017) https://doi.org/gg2c7t
DOI: 10.1007/978-3-319-55753-3_12

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

15. **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

16. **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

17. **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

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

19. **Estimating the complexity of hetnet traversal**
Daniel Himmelstein, Antoine Lizee

ThinkLab (2016-03-22) https://doi.org/gbr42x
DOI: 10.15363/thinklab.d187

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

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

22. **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

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

24. **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/

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

26. **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/gg2wr7
DOI: 10.5281/zenodo.1435834