DOI: 10.1093/gigascience/giad047

Q1: **[Copyeditor to Author]** AU: Please provide the short title for running head. **[Reply by Author]** Hetnet Connectivity Search Hetnet Connectivity Search

TECH NOTE

Hetnet connectivity search provides rapid insights into how **Q2:** **[Author to Editor]** The copyeditor changed "two" to "2" in the title. We don't like how this change reads and would instead like to delete two/2 from the title entirely. I have deleted "2" from the title here. Please ensure this change is made for any other occurrences of the title in the publication and Gigascience systems. ~~2~~ biomedical entities are related

Daniel S. Himmelstein[1,~~2~~], Michael Zietz[~~2~~1,3], Vincent Rubinetti[~~4,5~~1,4], Kyle Kloster[5,6], Benjamin J. Heil[7], Faisal Alquaddoomi[~~8,9~~4,8], Dongbo Hu[~~10~~9], David N. Nicholson[~~11~~], Yun Hao[~~12~~7], Blair D. Sullivan[~~13~~10], Michael W. Nagle[~~14,15~~1,12] and Casey S. Greene[~~16,17,1~~4,8,*]

[1]Department of Systems Pharmacology and Translational Therapeutics, University of Pennsylvania, Philadelphia, PA 19104, USA **Q3:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** Postal code has been added to the affiliation **Q4:** **[Copyeditor to Author]** AU: To check that we have your surnames correctly identified and tagged (e.g. for indexing), we have coloured green the names that we have assumed are surnames. If any of these are wrong, please let us know so that we can amend the tagging. **[Reply by Author]** surname identification is correct

~~[2]Department of Systems Pharmacology and Translational Therapeutics, University of Pennsylvania, Philadelphia, PA, USA~~

[2]Related Sciences, Denver, CO 80202, USA.

[3]Department of Biomedical Informatics, Columbia University, New York, NY 10032, ~~USA~~ USA **Q5:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** 10032

~~[4]Department of Systems Pharmacology and Translational Therapeutics, University of Pennsylvania, Philadelphia, PA, USA~~

[~~5~~4]Center for Health AI, University of Colorado School of Medicine, Aurora, CO 80045, ~~USA~~ USA **Q6:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** Postal code has been added to the affiliation

[5] Carbon, Inc., Redwood City, CA 94063, USA ,

[6]~~Carbon, Inc.;~~ Department of Computer Science, North Carolina State University, Raleigh, NC 27606, USA **Q7:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** Postal code has been added to the affiliation

[7]Genomics and Computational Biology Graduate Group, Perelman School of Medicine, University of Pennsylvania ~~, Philadelphia, PA, USA~~ , Philadelphia, PA 19104, USA **Q8:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** 19104

[8]Department of Biochemistry and Molecular Genetics, University of Colorado School of Medicine, Aurora, CO 80045, ~~USA~~ USA **Q9:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** Postal code has been added to the affiliation

~~[9]Center for Health AI, University of Colorado School of Medicine, Aurora, CO, USA~~

[~~10~~9]Department of Pathology, Perelman School of Medicine University of Pennsylvania, Philadelphia, PA 19104, USA **Q10:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** Postal code has been added to the affiliation

~~[11]Department of Systems Pharmacology and Translational Therapeutics, Perelman School of Medicine University of Pennsylvania, Philadelphia, PA, USA~~

~~[12]Genomics and Computational Biology Graduate Group, Perelman School of Medicine, University of Pennsylvania, Philadelphia, PA, USA~~

[~~13~~10]School of Computing, University of Utah, Salt Lake City, UT 84112, USA **Q11:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** Postal code has been added to the affiliation

[~~14~~11]Integrative Biology, Internal Medicine Research Unit, Worldwide Research, Development, and Medicine, Pfizer Inc, Cambridge, MA 02139, USA **Q12:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** Postal code has been added to the affiliation

[~~15~~12]~~Neurogenomics, Translational Sciences~~Human Biology Integration Foundation, ~~Neurology Business Group~~Deep Human Biology Learning, Eisai Inc., Cambridge, MA 02140, ~~USA~~ USA **Q13:** **[Copyeditor to Author]** AU: Please provide missing 'Postal code' for this affiliation **[Reply by Author]** Postal code has been added to the affiliation

~~[16]Department of Systems Pharmacology and Translational Therapeutics, University of Pennsylvania, Philadelphia, PA, USA~~

~~[17]Department of Biochemistry and Molecular Genetics, University of Colorado School of Medicine, Aurora, CO, USA~~

~~[18]Center for Health AI, University of Colorado School of Medicine, Aurora, CO, USA~~

[*]**Correspondence address**. Casey S. Greene, Email: casey.s.greene@cuanschutz.edu

Received: 7 January 2023; Revised: 14 April 2023; Accepted: 6 June 2023

Abstract

Background:

Hetnets, short for "heterogeneous networks," contain multiple node and relationship types and offer a way to encode biomedical knowledge. One such example, Hetionet, connects 11 types of nodes—including genes, diseases, drugs, pathways, and anatomical structures—with over 2 million edges of 24 types. Previous work has demonstrated that supervised machine learning methods applied to such networks can identify drug repurposing opportunities. However, a training set of known relationships does not exist for many types of node pairs, even when it would be useful to examine how nodes of those types are meaningfully connected. For example, users may be curious about not only how metformin is related to breast cancer but also how a given gene might be involved in insomnia.

Findings:

We developed a new procedure, termed *hetnet connectivity search*, that proposes important paths between any 2 nodes without requiring a supervised gold standard. The algorithm behind connectivity search identifies types of paths that occur more frequently than would be expected by chance (based on node degree alone). Several optimizations were required to precompute significant instances of node connectivity at the scale of large knowledge graphs.

Conclusion:

We implemented the method on Hetionet and provide an online interface at https://het.io/search. We provide an open-source implementation of these methods in our new Python package named hetmatpy.

Keywords:
knowledge graphs; hetnets; networks; connectivity; search; Hetionet; path counts; matrix multiplication; bioinformatics; algorithms

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 1 type of node and 1 type of edge. However, in many cases, it is necessary to be able to distinguish between different types of entities and relationships.

A *hetnet* (short for heterogeneous information network [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 describe more complex data accurately. Hetnets are particularly useful in biomedicine, where it is important to capture the conceptual distinctions between various entities, such as genes and diseases, and linkages, such as 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* refers to the type (e.g., compound), as opposed to a specific node/edge/path itself (e.g., acetaminophen).

One such network is Hetionet, which 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.

The initial application of Hetionet, named Project Rephetio, focused on drug repurposing [2]. The authors predicted the probability of drug efficacy for 209,168 compound–disease pairs. A supervised machine learning approach identified types of paths that occur more or less frequently between known treatments than nontreatments (Fig. 1B). To train the model, the authors created PharmacotherapyDB, a physician-curated catalog of 755 disease-modifying treatments [3].

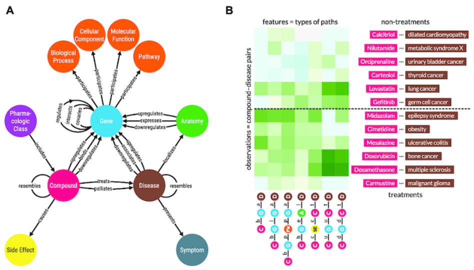

A
Cellular Component
Molecular Function
Biological Process
Pathway
Pharmacologic Class
Gene
Anatomy
Compound
Disease
Side Effect
Symptom
participates
regulates
interacts
covaries
upregulates
expresses
downregulates
includes
binds
resembles
treats
palliates
localizes
associates
causes
presents
B
features = types of paths
non-treatments
treatments
observations = compound–disease pairs
Calcitriol – dilated cardiomyopathy
Nilutamide – metabolic syndrome X
Orciprenaline – urinary bladder cancer
Carteolol – thyroid cancer
Lovastatin – lung cancer
Gefitinib – germ cell cancer
Midazolam – epilepsy syndrome
Cimetidine – obesity
Mesalazine – ulcerative colitis
Doxorubicin – bone cancer
Dexamethasone – multiple sclerosis
Carmustine – malignant glioma

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 bottom half of rows correspond to known treatments (i.e., positives), while the top half correspond to nontreatments (i.e., negatives under a *closed-world assumption*, not known to be treatments in PharmacotherapyDB). Here, an equal number of treatments and nontreatments are shown, but in reality, the problem is heavily imbalanced. Project Rephetio scaled models to assume a positive prevalence of 0.36% [2, 4]. Each column represents a metapath, labeled with its abbreviation.
Feature values are degree-weighted path counts (abbreviated DWPCs, transformed and standardized), which assess the connectivity along the specified metapath between the specific compound and disease. Green 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.

Project Rephetio successfully predicted treatments, including those under investigation by clinical trial. However, 2 challenges limit the applicability of Rephetio. 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 degree-weighted path count (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 develop Hetnet connectivity search, which defines a null distribution for DWPCs that accounts for degree and enables detecting meaningful hetnet connectivity without training labels.

Existing research into methods for determining whether 2 nodes are related primarily focuses on homogeneous networks (without type). Early approaches detected related nodes by measuring neighborhood overlap or path similarity between 2 nodes [5, 6]. These approaches predicted node relatedness with success. However, they are challenging to scale as a network grows in size or semantic richness (i.e., type) [5].

More recently, focus has shifted to graph embeddings to determine if 2 nodes are related, specifically in the context of knowledge graphs, which are often semantically rich and include type [7, 8, 9, 10, 11]. These types of methods involve mapping nodes and sometimes edges to dense vectors via neural network models [12, 13, 14], matrix factorization [15, 16], or translational distance models [17]. Bioteque creates node embeddings from the bipartite network of DWPCs for a given metapath [18]. Once these dense vectors have been produced, quantitative scores that measure node relatedness can be generated via a machine learning model [8, 19, 20] or by selected similarity metrics [7, 9, 21, 22, 23]. These approaches have been quite successful in determining node relatedness. Yet, they only state *whether* 2 nodes are related and fail to explain *why* 2 nodes are related.

Explaining why 2 nodes are related is a nontrivial task because approaches must output more than a simple similarity score. The first group of approaches output a list of ranked paths that are most relevant between 2 nodes [24, 25, 26]. For example, the FAIRY framework explains 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) [25]. ESPRESSO explains how 2 sets of nodes are related by returning subgraphs [27]. Other approaches such as MetaExp return important metapaths rather than paths but require some form of supervision [28, 29].

MechRepoNet is a hetnet containing 250,035 nodes across 9 metanodes and 9,652,116 edges across 68 metaedges [30]. The study trained a model using DWPCs as features to predict *compound–treats–disease* relationships, which was able to select 89 metapaths with positive regression coefficients. The authors also created DrugMechDB with a curated set of paths capturing known mechanisms of action for 123 compound–disease pairs [30]. Metapath coefficients were used to rank paths, using DrugMechDB as validation. The method generally performed well, although interpretability was challenging when "hundreds, or thousands of paths ranked above the mechanistic path in DrugMechDB" [30]. To address this issue, the study explored additional path filters, such as filtering for paths that traverse known drug targets, and dimensionality reduction by aggregating paths across intermediate nodes and summing the path weights. Refinements to path-scoring techniques might also be helpful solutions in this context.

Hetnet connectivity search explains how 2 nodes are related in an unsupervised manner that captures the semantic richness of edge type and returns results in the form of both metapaths and paths. Our open-source implementation, including for a query and visualization webserver, was designed with scalability and responsiveness in mind, allowing in-browser exploration.

Findings

Completing hetnet connectivity search involved advances on 3 fronts. We implemented new software for efficient matrix-based operations on hetnets. We developed strategies to efficiently calculate the desired connectivity score under the null. We designed and developed a web interface for easy access to the connectivity search approach.

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 matrix-based utilities for hetnets. Each metaedge is represented by a distinct adjacency matrix, which can be either a dense Numpy array or a sparse SciPy matrix (see HetMat architecture). Adjacency matrices are stored on disk and loaded in a lazy manner to help scale the software to hetnets that are too large to fit entirely in memory.

The primary focus of the package is to provide compute-optimized and memory-efficient implementations of path-counting algorithms. Specifically, the package supports computing DWPCs, which can be done efficiently using matrix multiplication but require complex adjustments to avoid counting paths with duplicate nodes (i.e., to filter walks that are not paths, see DWPC matrix multiplication algorithms). The package can reuse existing path count computations that span segments of a longer metapath. The package also supports generating null distributions for DWPCs derived from permuted networks (see "Degree-grouping of node pairs"). Since this approach generates too many permuted DWPC values to store on disk, our implementation retains summary statistics for each degree group that allow computation of a gamma-hurdle distribution from which null DWPC *P* values can be generated.

DWPC null distribution

To assess connectivity between a source and target node, we use the DWPC 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 [31].

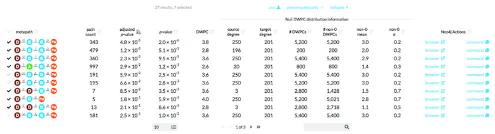
Previously, we had no technique for detecting whether a DWPC value was exceptional. One possibility is to evaluate the DWPCs for all pairs of nodes and select the top scores (e.g., the top 5% of DWPCs). Another possibility is to pick a transformed DWPC score as a cutoff. The shortcomings of these methods are 2-fold. First, neither the percentile nor absolute value of a DWPC has inherent meaning. To select transformed DWPCs greater than 3.5, 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 due to their degree (4).

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. Our null model is based on DWPCs generated from permuted networks, where edges have been randomized in a degree-preserving manner (see "Permuted hetnets").

By tailoring the null distribution for a DWPC to the degree of its source and target node (see "Degree-grouping of node pairs"), we account for degree effects when determining the significance of a DWPC. To improve the accuracy of DWPC *P* values, we use fit a gamma-hurdle distribution to the null DWPCs. In rare cases, there are insufficient nonzero null DWPCs to fit the gamma portion of the null distribution. In these cases, we fall back to an empirical calculation as described in "Empirical DWPC *P* values."

Enriched metapaths

For each of the 2,205 metapaths in Hetionet v1.0 with length ≤3, we computed DWPCs for all node pairs and their corresponding null distributions (see "DWPC and null distribution computation"). We store the most significant DWPCs as described in "Prioritizing enriched metapaths for database storage," which appear as the "precomputed" rows in the webapp metapath table (Figs. 3B and 2). DWPCs that are not retained by the database can be regenerated on the fly. This design allows us to immediately provide users with the metapaths that are most enriched between 2 query nodes while still allowing on-demand access to the full metrics for all metapaths with length ≤3.


27 results, 7 selected   .csv   precomputed only ✓   collapse

	metapath	path count	adjusted p-value	p-value	DWPC	source degree	target degree	# DWPCs	# non-0 DWPCs	non-0 mean	non-0 σ	Neo4j Actions	
						Null DWPC distribution information							
✓	D G G PW	343	4.8×10^{-5}	2.0×10^{-6}	3.8	250	201	5,200	5,200	3.0	0.2	browser	command
✓	D G G PW	479	1.2×10^{-3}	5.1×10^{-5}	2.8	196	201	200	200	2.0	0.2	browser	command
✓	D G G PW	360	2.3×10^{-3}	9.5×10^{-5}	3.6	250	201	5,400	5,400	2.9	0.2	browser	command
✓	D A G PW	997	2.9×10^{-2}	1.2×10^{-3}	2.6	20	201	800	800	1.4	0.3	browser	command
	D G G PW	191	5.9×10^{-2}	2.5×10^{-3}	3.6	250	201	5,400	5,400	3.0	0.2	browser	command
	D G G PW	195	6.6×10^{-2}	2.8×10^{-3}	3.6	250	201	5,200	5,200	3.0	0.2	browser	command
✓	D D G PW	7	8.5×10^{-2}	3.5×10^{-3}	3.6	3	201	2,800	1,428	1.5	0.7	browser	command
✓	D G PW	5	1.8×10^{-1}	5.9×10^{-2}	4.0	250	201	5,200	5,021	2.8	0.7	browser	command
✓	D D G PW	13	2.1×10^{-1}	8.6×10^{-3}	2.8	3	201	2,800	2,718	1.1	0.5	browser	command
	D G G PW	181	2.5×10^{-1}	1.0×10^{-2}	3.6	250	201	5,400	5,400	3.0	0.2	browser	command

10   1 of 3

Figure 2: Expanded metapath details from the connectivity search webapp. This is the expanded view of the metapath table in Figure 3B showing enriched metapaths between Alzheimer's disease and the circadian rhythm pathway.

Fig. 2 shows the information used to compute the *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 the "# of DWPCs" column that were nonzero. Nonzero DWPCs indicate at least 1 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

In addition to knowing which metapaths are enriched between 2 query nodes, it is helpful to see the specific paths that contribute highly to such enrichment. Since the DWPC is a summation of a path metric (called the path degree product), it is straightforward to calculate the proportion of a DWPC attributable to an individual path. The webapp allows users to select a metapath to populate a table of the corresponding paths. These paths are generated on the fly through a Cypher query to the Hetionet Neo4j database.

It is desirable to have a consolidated view of paths across multiple metapaths. Therefore, we calculate a *path score* heuristic, which can be used to compare the importance of paths between metapaths. The path score equals the proportion of the DWPC contributed by a path multiplied by the magnitude of the DWPC's *P* value ($-\log_{10}(P)$). To illustrate, the paths webapp panel includes the following information (Fig. 3C):

- **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 percentage 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.

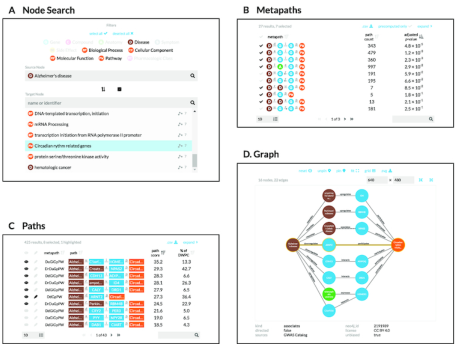


Figure 3: Using the connectivity search webapp to explore the pathophysiology of Alzheimer's disease. This figure shows an example user workflow for https://het.io/search/.
(A) The user selects 2 nodes. Here, the user is interested in Alzheimer's disease, so selects this as the source node. The user 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 (limited to 100 paths for each metapath). 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 an association between Creutzfeldt–Jakob disease and *NPAS2* was detected by ~~GWAS~~genome-wide association study. **Q14: [Copy Editor Freelancer to Author]** Author: Please define GWAS. **[Reply by Author]** removed acronym.

Hetio website and connectivity search webapp

We revamped the website hosted at https://het.io to serve as a unified home for this study and the hetnet-related research that preceded it. The website provides the connectivity search webapp running over the hetio network and several other interactive apps for prior projects. It also includes high-level information on hetnets and Hetionet, citation and contact details, links to supporting studies and software, downloads and exploration of Hetionet data, and related media.

We created the connectivity search webapp available at https://het.io/search/. The tool is free to use, without any login or authentication. The app allows users to quickly explore how any 2 nodes in Hetionet v1.0 might be related. The workflow accepts 1 or more nodes as input and shows 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 (Fig. 3A). The webapp returns metapaths, scored by whether they occurred more than expected based on network degree (Fig. 3B). Users can proceed by requesting the specific paths for each metapath, which are placed in a unified table sorted according to their path score (Fig. 3C). Finally, the webapp produces publication-ready visualizations containing user-selected paths (Fig. 3D).

Discussion

In this study, we introduce a search engine for hetnet connectivity between 2 nodes that returns results in real time. An interactive webapp helps users explore node connectivity by ranking metapaths and paths while visualizing multiple paths in a subgraph.

We made several methodological contributions to support this application. 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. For many queries, a large number of paths are returned. Interpretation of large lists is difficult. Therefore, the dimensionality of results could be reduced by aggregating path scores across intermediate nodes or edges [32].

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 detected as informative. For example, if a *CbGpPWpG* metapath is deemed informative, 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 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.

This work focuses 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 across 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 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 predefined gene sets to match against. Some users might be interested 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, and so on.

Our work is not without limitations. The final application relies on multiple databases and cached computations specific to Hetionet v1.0. Despite striving for a modular architecture, generating an equivalent search webapp for a different hetnet would require adaptation due to the many data sources 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

Hetionet

We used the hetionet knowledge graph to demonstrate connectivity search. 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 (Fig. 1A).

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

Metanode	Abbreviation	Nodes	Description
Anatomy	A	402	Anatomical structures, excluding structures that are known not to be found in humans. From Uberon.
Biological Process	BP	11,381	Larger processes or biological programs accomplished by multiple molecular activities. From Gene Ontology.
Cellular Component	CC	1,391	The locations relative to cellular structures in which a gene product performs a function. From Gene Ontology.
Compound	C	1,552	Approved small-molecule compounds with documented chemical structures. From DrugBank.
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	20,945	Protein-coding human genes. From Entrez Gene.
Molecular Function	MF	2,884	Activities that occur at the molecular level, such as "catalysis" or "transport." From Gene Ontology.
Pathway	PW	1,822	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	5,734	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.

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 [33], while Hetionet includes only 137 diseases [34]. 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 [2]. 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 [35]. Other integrative resources, some still under development, include Wikidata [36], SemMedDB [37, 38, 39], SPOKE [40], and RTX-KG2c [41].

HetMat architecture

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 reside 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 [2, 42], but we renamed it het**net**py for better disambiguation from het**mat**py.
2. A nodes directory containing 1 file per node type (metanode) that defines each node. Currently, .tsv files in which each row represents a node are supported.
3. An edges directory containing 1 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~~ path-counts **Q15:** **[Author to Editor]** Here path-counts refers to the name of a directory in the HetMat file structure, so I don't think we want to remove the hyphen. Ideally this could be formatted as inline code / preformatted text to make this clear, but I don't think PXE5 has that option. and permutations, store data generated from the HetMat. By using consistent paths for generated data, we avoid recomputing data that already exist 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 preexisting 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 2 implementations for computing DWPCs. The first is a pure Python implementation available in the hetnetpy.pathtools.DWPC function [42]. The second uses a Cypher query, prepared by hetnetpy.neo4j.construct_dwpc_query, that is executed by the Neo4j database [2,43]. Both of these implementations require traversing all paths between the source and target node. Hence, they are computationally cumbersome despite optimizations [44].

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 or walks) [45]. We developed a suite of algorithms to compute true path counts and DWPCs using matrix multiplication that benefits from the speed advantages of only counting paths.

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 nonrepeated metanodes (e.g., *G* in *DrDaGaDrD*) require no special treatment and are included in DWPC with 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 efficiently evaluate DWPC 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 also allowed us to implement a caching strategy that improved speed by avoiding duplicate DWPC computations. In summary, the functionality we developed resulted in more than a 175-fold reduction in compute time, allowing us to compute millions of DWPC values across Hetionet [46].

Details of matrix DWPC implementation

DWPC computation requires us to remove all duplicate nodes from paths. We used 3 repeat patterns as the building blocks for DWPC computation: short repeats (AAA), nested repeats (BAAB), and overlapping repeats (BABA). Let $D(XwXyZ)$ denote the DWPC matrix for metapath *XwXyZ*. Under this notation, $D(XyZ)$ is the degree-weighted (bi)adjacency matrix for metaedge *XyZ*. Additionally, let $diag(A)$ 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.

$$D\left(XaXbX\right) = D\left(XaX\right) D\left(XbX\right) - diag\left(D\left(XaX\right) D\left(XbX\right)\right)$$

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

$$D\left(XaYbYcX\right) = D\left(XaY\right) D\left(YbY\right) D\left(YcX\right) - diag$$

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

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

Most paths of length 6—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 4 repeat). Mayers et al. [47] developed an alternative approximation, which subtracts the main diagonal at every occurrence of the first repeated metanode. Our matrix methods were validated against the existing Python and Cypher implementations in the hetnetpy package that rely on explicit path enumeration.

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 [48]. XSwap randomizes edges while preserving node degree. Therefore, it is 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 [2, 49, 50].

Project Rephetio created 5 permuted hetnets [2, 49], 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. Permutations 001–005 were those generated by Project Rephetio, while permutations 006–200 were generated by this study. For the newly generated permutations, we attempted 10 times the number of swaps as edges for a given metaedge, which is the default multiplier set by the hetnetpy permute_graph function. 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 [50].

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 [50, 51]. 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 also on all permuted DWPCs for the source-degree–metapath–target-degree combination.

The “# DWPCs” column in Fig. 2 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 most needed.

One final benefit of degree-grouping is that it reduces the disk space required to store null DWPC summary statistics. For example, with 20,945 genes in Hetionet v1.0, there exist 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.

Fig. 4 shows how various aspects of DWPCs vary by degree group. The rows display the following metrics of the DWPC distribution for all node pairs in a given degree-group:

- **# Nonzero DWPCs**: The number of nonzero DWPCs values (on average per network to enable comparison).
- **% Nonzero DWPCs**: Of the total number of DWPCs, the percentage that is nonzero. As node degrees increase, the chance of node pairs having at least 1 path, and hence a nonzero DWPC, greatly increases.
- **Mean DWPC**: The average value of all DWPCs, including zeros.
- **Mean Nonzero DWPC**: The average value of nonzero DWPCs.
- **Std Dev Nonzero DWPC**: The standard deviation of nonzero DWPCs.
- **Gamma Model β**: The β parameter of the gamma model fit on nonzero DWPCs. Note that the gamma model is only fit on permuted network DWPCs to estimate a null distribution for the unpermuted network DWPCs. Since this parameter varies with source and target node degree, it is important to fit a separate gamma model for each degree group.

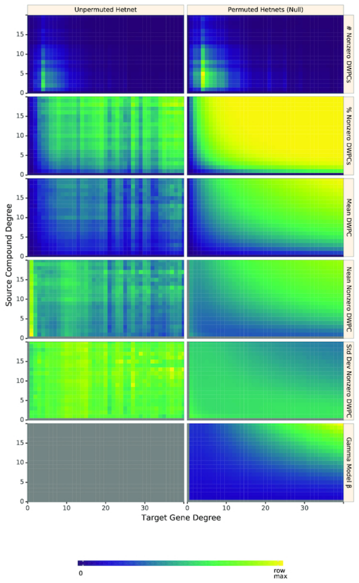


Figure 4: Path-based metrics vary by node degree and network permutation status. Each row shows a different metric of the DWPC distribution for the *CbGpPWpG* metapath—traversing Compound–binds–Gene–participates–Pathway–participates–Gene, selected for illustrative purposes. Metrics are computed for degree-groups, which is a specific pair of source degree (in this case, the source compound's count of CbG edges) and target degree (in this case, the target gene's count of GpPW edges). Metrics are reported for the unpermuted hetnet on the left and for the 200 permuted hetnets on the right. Hence, each cell on the right summarizes 200 times the number of DWPCs as the corresponding cell on the left. The color map is row normalized, such that its intensity peaks for the maximum value of each metric across the unpermuted and permuted values. Gray indicates null values.

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 is often zero, indicating that the source and target nodes are not connected along the metapath in the permuted network. Second, we observed that nonzero 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 estimated the probability of observing a permuted DWPC value greater than DWPC computed in the unpermuted network, akin to a 1-tailed *P* value. These quasi-significance scores ("*P* values") allow us to identify outlier node pairs at the metapath level (see examples in Fig. 5).

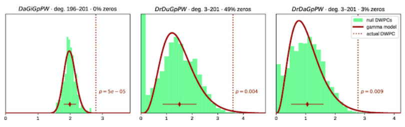


Figure 5: From null distribution to *P* value for DWPCs. Null DWPC distributions are shown for 3 metapaths between Alzheimer's disease and the circadian rhythm pathway, selected from Fig. 2. For each metapath, null DWPCs are computed on 200 permuted hetnets and grouped according to source–target degree. 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 gamma-hurdle distribution is defined over the support [0, ∞). The probability of a draw, *X*, from the gamma-hurdle distribution is given as follows:

$$P(X = 0) = 1 - \lambda$$

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

We estimate all 3 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.

$$\hat{\lambda} = \frac{n}{N}$$

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

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

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 do not 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. 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 [42]. When $w = 0$, the DWPC is equivalent to the path count. Previously, we found $w = 0.4$ was optimal for predicting disease-associated genes [42]. 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 requirements 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 2-dimensional 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 [52]. 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 [52]. Due to degree-grouping, null DWPC 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

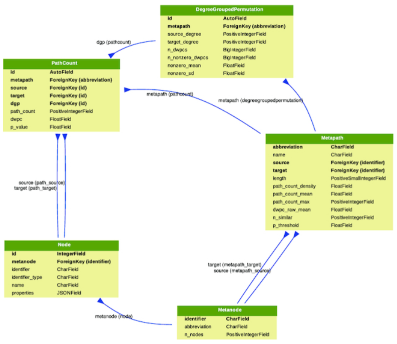
When a user applies hetnet connectivity search to identify enriched metapaths between 2 nodes, many metapaths are evaluated for significance. Due to multiple testing of many DWPCs, low *P* values are likely to arise by chance. Therefore, we devised a multiple-testing correction.

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 Fig. 2 as an example, the DdGpPW *P* value of 5.9% was adjusted to 17.8% (multiplied by a factor of 3).

Bonferroni controls family-wise 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 [53], 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 Fig. 6) enables the connectivity search webapp to provide users with enriched metapaths between query nodes in real time. 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 a 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 are 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.



DegreeGroupedPermutation	
id	**AutoField**
metapath	**ForeignKey (abbreviation)**
source_degree	PositiveIntegerField
target_degree	PositiveIntegerField
n_dwpcs	BigIntegerField
n_nonzero_dwpcs	BigIntegerField
nonzero_mean	FloatField
nonzero_sd	FloatField

PathCount	
id	**AutoField**
metapath	**ForeignKey (abbreviation)**
source	**ForeignKey (id)**
target	**ForeignKey (id)**
dgp	**ForeignKey (id)**
path_count	PositiveIntegerField
dwpc	FloatField
p_value	FloatField

Metapath	
abbreviation	**CharField**
name	CharField
source	**ForeignKey (identifier)**
target	**ForeignKey (identifier)**
length	PositiveSmallIntegerField
path_count_density	FloatField
path_count_mean	FloatField
path_count_max	PositiveIntegerField
dwpc_raw_mean	FloatField
n_similar	PositiveIntegerField
p_threshold	FloatField

Node	
id	**IntegerField**
metanode	**ForeignKey (identifier)**
identifier	CharField
identifier_type	CharField
name	CharField
properties	JSONField

Metanode	
identifier	**CharField**
abbreviation	CharField
n_nodes	PositiveIntegerField

Figure 6: 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).

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 2 query nodes. For some node pairs, the on-the-fly computation is quick (less than a second). Other times, computing DWPCs for all metapaths might take more than a minute.

Backend database and 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.

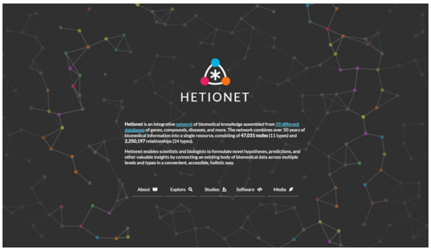
We define the database schema using Django's object-relational mapping framework (Fig. 6). We import the data into a PostgreSQL database. Populating the database for all 2,205 metapaths up to length 3 was a prolonged operation, taking over 3 days. The majority of the time is spent populating the DegreeGroupedPermutation (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.

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 preexisting Hetionet v1.0 Neo4j graph database. They were designed to power the hetnet connectivity search webapp but are also available for general research use.

Frontend

Hetio website

We created a static website to serve as the home for the Hetio organization using Jekyll hosted on GitHub Pages (Fig. 7). The source code is available in the het.io repository. To make a change to the website, an author simply commits the changes (either directly or through a pull request) to the repository's gh-pages branch, and GitHub automatically recompiles the website and hosts the resulting files at the provided custom domain URL.



Hetionet is an integrative network of biomedical knowledge assembled from 29 different databases of genes, compounds, diseases, and more. The network combines over 50 years of biomedical information into a single resource, consisting of **47,031 nodes** (11 types) and **2,250,197** relationships (24 types).

Hetionet enables scientists and biologists to formulate novel hypotheses, predictions, and other valuable insights by connecting an existing body of biomedical data across multiple levels and types in a convenient, accessible, holistic way.

Figure 7: Homepage of the Hetio website. The redesigned homepage provides a succinct overview of what Hetionet consists of and what its purpose is.

Webapps

We developed the connectivity search app as a single-page, standalone application using React and associated tools. The source code is available in the connectivity-search-frontend repository.

Since the rest of the overarching Hetio website was mostly noninteractive content, it was appropriate to construct the bulk of the website in simple static formats like Markdown and HTML using Jekyll and leave React for implementing the sections of the site that required more complex behavior.

We used React's own create-react-app command-line tool to generate a boilerplate for the app. This simplified setup, testing, and building pipelines, bypassing time-consuming configuration of things like Webpack and linters. Some configuration was necessary to produce nonhashed, consistently named output files like index.js that could be easily and reliably referenced by and embedded into the Hetio Jekyll website.

For authoring components, we used React's traditional class syntax. At the time of authoring the app, React Hooks were still nascent, and thus the simpler and less-verbose functional syntax was not viable.

While writing this application, we also elected to rewrite the preexisting Rephetio and disease-associated genes apps in the same manner. We created a custom package of React components and utility functions that could be shared across the multiple interactive apps on the website. The package is located at and can be installed from the frontend-components repository. The package consists of interface “components” (building blocks) like buttons and sortable/searchable/paginated tables as well as utility functions for formatting data and debugging. Each of the interactive apps imports this package to reduce code repetition and to enforce a consistent style and behavior across the website.

For managing state in the connectivity search app, we used the Redux library. Redux was chosen over vanilla React or other state management libraries since:

1 The state in this app was very “global,” meaning most of it was needed by a lot of different parts of the app. Redux provides a convenient global “store” of state that is easily accessible to any component, avoiding the “prop-drilling” phenomenon.
2 The structure of the state is nested and complex. Redux's “reducer” approach makes it cleaner to modify this type of data.
3 Redux's approach to immutable state that is updated by actions and pure functions makes the application easier to debug. It is easy to get a clear timeline of how and when the state changed and what changed it.

To create the graph visualization at the bottom of the app, the D3 library was used. D3 satisfied several core requirements:

1 ~~SVG~~ Scalable Vector Graphics **Q16:** **[Copy Editor Freelancer to Author]** Author: Please define SVG. **[Reply by Author]** replaced this file format abbreviation with the verbose name implementation for high-resolution, publication-ready figures
2 Force-directed layout for untangling nodes
3 Pinnable nodes and other physics customizations
4 Customizable node and edge drag/hover/select behavior
5 Intuitive pan/zoom view that worked on desktop and mobile
6 Node and edge appearances that were completely customizable for alignment, text wrapping, color, outlines, fonts, arrowheads, and noncolliding coincident edges

Visual design

A limited palette of colors was chosen to represent the different types of nodes (metanodes) in the Hetionet knowledge graph. These colors are listed and programmatically accessible in the hetionet repository under/describe/styles.json.

At the time of developing connectivity search, Hetionet already had an established palette of colors (from Project Rephetio). To avoid confusion, we were careful to keep the general hue of each metanode color the same for backward compatibility (e.g., genes stayed generally blue, diseases stayed generally brown). In this way, this palette selection was more of a modernization/refresh. For cohesiveness, accessibility, and aesthetic appeal, we used the professionally curated Material Design palette as a source for the specific color values.

The palette is now used in all Hetio-related applications and materials. This is not just to maintain a consistent look and feel across the Hetio organization but to convey clear and precise meaning. For example, the colors used in the metagraph in Fig. 1A are exactly the same colors and thus represent the same types of entities, as in *any part of the connectivity search app* (Fig. 3).

Colors in the palette are also used in the Hetio logo (seen in Fig. 7) and other miscellaneous logos and iconography across the website, to establish an identifiable brand for the Hetio organization as a whole.

Real-time 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 opportunity for other contributors to review changes before they are officially accepted. For example, in greenelab/connectivity-search-analyses#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 [54]. 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 and data availability

Hetio is a superset/collection of hetnet-related research, tools, and datasets that, most notably, includes the Hetionet project itself and the connectivity search tool that are the focus of this article. Most of the Hetio resources and projects can be found under the Hetio GitHub organization, with others being available under the Greene Lab GitHub organization, one of the collaborating groups. Information about Hetio is also displayed and disseminated at https://het.io, as noted in the Hetio Website section.

Availability of Supporting Source Code and Requirements

- Project name: Hetnet Connectivity Search
- Project homepage: https://het.io/search/
- Operating systems: Platform independent
- Programming language: Python, Javascript, Cypher
- Other requirements: Refer to specific repositories below for their respective dependency configuration files
- License: Refer to specific repositories below, but generally software is released under BSD, figures and documentation under CC BY, and data under CC0.
- RRID: SCR_023630
- biotools ID: connectivity-search

This study primarily involves the following GitHub repositories:

- greenelab/connectivity-search-manuscript [55]: Source code for this manuscript. Best place for general comments or questions. CC BY 4.0 License.
- greenelab/connectivity-search-analyses [56]: 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 [57]: Source code for the connectivity search database and API. BSD 3-Clause License.
- greenelab/connectivity-search-frontend [58]: Source code for the connectivity search webapp. BSD 3-Clause License.
- hetio/hetmatpy [59]: Python package for matrix storage and operations on hetnets. Released on PyPI. BSD 2-Clause Plus Patent License. Registered at biotools:hetmatpy and RRID:SCR_023409.
- hetio/hetnetpy [60]: 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 [61]: Preexisiting data repository for Hetionet, including the public Neo4j instance and HetMat archives. CC0 1.0 License.
- hetio/het.io [62]: Preexisiting source code for the https://het.io/website. CC BY 4.0 License.

Abbreviations

Arnt: aryl hydrocarbon receptor nuclear translocator protein; ARNT2: aryl hydrocarbon receptor nuclear translocator 2; DWPC: degree-weighted path count; LFS: large file storage; NPAS2: neuronal PAS domain protein 2; PAS: Per-Arnt-Sim; Per: period circadian protein; Sim: single-minded protein.

Funding

D.S.H., B.J.H., D.H., and D.N.N. were funded by The Gordon and Betty Moore Foundation (GBMF4552). D.S.H. and C.S.G. were funded by Pfizer Worldwide Research, Development, and Medical. K.K. was funded by The Gordon and Betty Moore Foundation (GBMF4560). D.N.N. was funded by The National Institutes of Health (T32 HG000046). C.S.G. was funded by the National Human Genome Research Institute (R01 HG010067), the National Cancer Institute (R01 CA237170), the Gordon and Betty Moore Foundation (GBMF 4552), and the Eunice Kennedy Shriver National Institute of Child Health and Human Development (R01 HD109765). The funders had no role in the study design, data analysis and interpretation, or writing of the manuscript.

Data Availability

An archival copy of the code and supporting data is available via the *GigaScience* repository [63]. The connectivity-search-analyses 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 [52].

Competing Interests

This work was supported, in part, by Pfizer Worldwide Research, Development, and Medical.

Authors’ Contributions

Author contributions are noted here according to CRediT (Contributor Roles Taxonomy). Conceptualization by D.S.H., M.Z., K.K., B.D.S., and C.S.G. Data curation by D.S.H., M.Z., V.R., and D.N.N. Formal analysis by D.S.H., M.Z., K.K., and B.J.H. Funding acquisition by D.S.H., B.D.S., and C.S.G. Investigation by D.S.H., M.Z., B.J.H., Y.H., M.W.N., and C.S.G. Methodology by D.S.H., M.Z., V.R., K.K., B.J.H., and C.S.G. Project administration by D.S.H. and C.S.G. Resources by D.S.H., F.S.A., D.H., M.W.N., and C.S.G. Software by D.S.H., M.Z., V.R., B.J.H., F.S.A., and D.H. Supervision by D.S.H., B.D.S., and C.S.G. Visualization by D.S.H., M.Z., V.R., and Y.H. Writing—original draft by D.S.H., M.Z., V.R., D.N.N., and C.S.G. Writing—review & editing by D.S.H., M.Z., V.R., D.H., B.D.S., and C.S.G. Validation by B.J.H. **Q17:** **[Copyeditor to Author]** AU: Permissions: Permission to reproduce any third party material in your paper should have been obtained prior to acceptance. If your paper contains figures or text that require permission to reproduce, please confirm that you have obtained all relevant permissions and that the correct permission text has been used as required by the copyright holders. Please contact jnls.author.support@oup.com if you have any questions regarding permissions. **[Reply by Author]** Not relevant. **Q18:** **[Copyeditor to Author]** AU: Please check that funding is recorded in a separate funding section if applicable. Use the full official names of any funding bodies, and include any grant numbers. **[Reply by Author]** See funding above **Q19:**

[Copyeditor to Author] AU: Journal policy requires authors to provide a data availability statement in their manuscript. Please confirm that this statement is included in your manuscript, or provide one if you have not already, and that any required links or identifiers for your data are present in the manuscript as described or provide edits with the required information. **[Reply by Author]** See above.

References

1. Himmelstein D, Greene C, Baranzini S. *Renaming 'Heterogeneous Networks' to a More Concise and Catchy Term*. ThinkLab; 2015. **Q20: [Copy Editor Freelancer to Author]** Author: Please provide the city of publication for Refs. 1, 3, 4, 31, 32, 34, 43, 44, 45, 49, 51. **[Reply by Author]** Refs. 1, 3, 4, 31, 32, 34, 43, 44, 45, 49, 51 refer to the Thinklab references. Thinklab was an online discussion forum where much of the foundational research for this study was performed. Since it is online, there is no city of publication. I think the most critical thing is for the Thinklab references to show their hyperlinked DOI, such that users can easily be directed to the Thinklab content. **Q21: [Author to Editor]** For the references, could the copyeditor take another look? The most important thing for us is that each reference includes a persistent identifier (such as a DOI or URL) that can both identify and retrieve the cited work. It is not clear exactly how the references in PXE5 get rendered to the formatted references in the final output PDF and HTML. I am not sure how to add DOIs and the DOIs that exist are embedded in a special way that I cannot figure out how to edit. Persistent identifiers are available for all of our references at https://greenelab.github.io/connectivity-search-manuscript/v/53fc9f3862b9b38ebd463edce6cda72ab1cb64a7/#references. Raw reference metadata is also available in CSL JSON format at https://github.com/greenelab/connectivity-search-manuscript/blob/output/references.json. Could the copyeditor please look over the references again and take the lead on fixing any additional issues including missing persistent identifiers? Thanks!
2. Himmelstein DS, Lizee A, Hessler C, et al. Systematic integration of biomedical knowledge prioritizes drugs for repurposing. 6 e26726 eLife 2017~~.~~ **Q22: [Copy Editor Freelancer to Author]** Author: Please provide the volume number and page range for Refs. 2 and 15. **[Reply by Author]** Added volume and pages info to references 2 and 15. They are both online only journals, so I used the article number/locator for the page number. Please double check.
3. Himmelstein D. *Announcing PharmacotherapyDB: The Open Catalog of Drug Therapies for Disease*. ThinkLab; 2016.
4. Himmelstein D. *Our Hetnet Edge Prediction Methodology: The Modeling Framework for Project Rephetio*. ThinkLab; 2016.
5. Liben-Nowell D, Kleinberg J. The link-prediction problem for social networks. J Am Soc Inf Sci 2007;58:1019–31.
6. Lü L, Zhou T. Link prediction in complex networks: a survey. Physica A 2011;390:1150–70.
7. Yang K, Wang N, Liu G, et al. Heterogeneous network embedding for identifying symptom candidate genes. J Am Med Inform Assoc 2018;5:1452–9.
8. Abdelaziz I, Fokoue A, Hassanzadeh O, et al. Large-scale structural and textual similarity-based mining of knowledge graph to predict drug–drug interactions. J Web Semantics 2017;4:104–17.
9. Gong F, Wang M, Wang H, et al. SMR: medical knowledge graph embedding for safe medicine recommendation. Big Data Res 2021;3:100174.
10. Ali M, Berrendorf M, Hoyt CT, et al. PyKEEN 1.0: a Python library for training and evaluating knowledge graph embeddings. J Machine Learn Res 2021;22:1–6.
11. Bonner S, Barrett IP, Ye C, et al. Understanding the performance of knowledge graph embeddings in drug discovery. Artif Intell Life Sci 2022;2:100036.
12. Grover A, Leskovec J. node2vec: scalable feature learning for networks. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM; 2016. **Q23: [Copy Editor Freelancer to Author]** Author: Please provide the editor(s) if applicable and the city of publication for the following references: 12, 13, 19, 25, 26, 27, 28, 38, and 48. **[Reply by Author]** In general, the editors are not applicable for these references. The editorial / copyedit team is welcome to visit the associated DOIs and add the editors if that information can be found and is deemed important. I am not sure however why the editor is being requested for some journal articles but not others.
13. Dong Y, Chawla NV, Swami A. metapath2vec: scalable representation learning for heterogeneous networks. In: *KDD '17: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM; 2017.
14. Gao Z, Fu G, Ouyang C, et al. edge2vec: representation learning using edge semantics for biomedical knowledge discovery. BMC Bioinf 2019;20. **Q24: [Copy Editor Freelancer to Author]** Author: Please provide the page range for the following references: 14, 18, 22, 35, 36, 39, 41, and 53. **[Reply by Author]** These references are mostly for online articles in electronic journals. The page number therefore has no relevance. Our submission provides copious links (DOIs, PumMed IDs, PubMed Central IDs). See https://greenelab.github.io/connectivity-search-manuscript/v/53fc9f3862b9b38ebd463edce6cda72ab1cb64a7/#references. If the editorial team can find page numbers for these online publications, they are welcome to add them.
15. Paliwal S, de Giorgio A, Neil D, et al. Preclinical validation of therapeutic targets predicted by tensor factorization on heterogeneous graphs. Sci Rep 2020;10 18250 ~~.~~
16. Zitnik M, Zupan B. Data fusion by matrix factorization. IEEE Trans Pattern Anal Mach Intell 2015;37:41–53.
17. Bordes A, Usunier N, Garcia-Durán A, et al. Translating embeddings for modeling multi-relational data. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems*. Red Hook, NY: Curran Associates Inc.; 2013:2787–95. **Q25: [Copy Editor Freelancer to Author]** Author: Please provide editor(s) if applicable. **[Reply by Author]** N/A
18. Fernández-Torras A, Duran-Frigola M, Bertoni M, et al. Integrating and formatting biomedical data as pre-calculated knowledge graph embeddings in the Bioteque. Nat Commun 2022;3.
19. Wang X, Gong Y, Yi J, et al. Predicting gene-disease associations from the heterogeneous network using graph embedding. In: *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE; 2019.
20. Li L, Wang P, Wang Y, et al. A method to learn embedding of a probabilistic medical knowledge graph: algorithm development. JMIR Med Inform 2020;8:e17645.
21. Alshahrani M, Hoehndorf R. Semantic Disease Gene Embeddings (SmuDGE): phenotype-based disease gene prioritization without phenotypes. Bioinformatics 2018;4:i901–7.
22. Xu B, Liu Y, Yu S, et al. A network embedding model for pathogenic genes prediction by multi-path random walking on heterogeneous network. BMC Med Genomics 2019;12.
23. Zong N, Kim H, Ngo V, et al. Deep mining heterogeneous networks of biomedical linked data to predict novel drug–target associations. In: Wren J, ed. *Bioinformatics*. Oxford: Oxford University Press; 2017.
24. Pirrò G. Explaining and suggesting relatedness in knowledge graphs. In: *The Semantic Web—ISWC 2015*. Springer International Publishing. **Q26: [Copy Editor Freelancer to Author]** Author: Please provide editor(s) and year of publication. **[Reply by Author]** I did not see the possibility of adding the year. The publication year is 2015. This is a conference preceeding where the conference had 11 editors.
25. Ghazimatin A, Saha Roy R, Weikum G. FAIRY: a framework for understanding relationships between users' actions and their social feeds. In: *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. ACM; 2019.
26. Wang Y, Carman MJ, Li Y-F. Using knowledge graphs to explain entity co-occurrence in Twitter. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM; 2017.
27. Seufert S, Berberich K, Bedathur SJ, et al. ESPRESSO: explaining relationships between entity sets. In: *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. ACM; 2016.
28. Behrens F, Aghaei F, Müller E, et al. MetaExp: interactive explanation and exploration of large knowledge graphs.In: *WWW '18: Companion Proceedings of the The Web Conference 2018*. ACM; 2018.
29. Meng C, Cheng R, Maniu S, et al. Discovering meta-paths in large heterogeneous information networks. In: *Proceedings of the 24th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee*. 2015. **Q27: [Copy Editor Freelancer to Author]** Author: Please provide editor(s) if applicable, publisher, and city of publication. **[Reply by Author]** In general these details are not applicable. All references are to online content that can be fully retrieved and identified via the provided DOIs.
30. Mayers M, Tu R, Steinecke D, et al. Design and application of a knowledge network for automatic prioritization of drug mechanisms. In: Wren J, ed. *Bioinformatics*. Oxford: Oxford University Press; 2022.
31. Himmelstein D, Khankhanian P, Lizee A. *Transforming DWPCs for Hetnet Edge Prediction*. ThinkLab; 2016.
32. Himmelstein D. *Decomposing the DWPC to Assess Intermediate Node or Edge Contributions*. ThinkLab; 2016.
33. Schriml LM, Mitraka E, Munro J, et al. Human Disease Ontology 2018 update: classification, content and workflow expansion. Nucleic Acids Res 2019;7:D955–62.
34. Himmelstein D, Li TS. *Unifying Disease Vocabularies*. ThinkLab; 2015.
35. Bonner S, Barrett IP, Ye C, et al. A review of biomedical datasets relating to drug discovery: a knowledge graph perspective. Briefings Bioinf 2022;3.
36. Waagmeester A, Stupp G, Burgstaller-Muehlbacher S, et al. Wikidata as a knowledge graph for the life sciences. eLife 2020;9.
37. Kilicoglu H, Shin D, Fiszman M, et al. SemMedDB: a PubMed-scale repository of biomedical semantic predications. Bioinformatics 2012;8:3158–60.
38. Cong Q, Feng Z, Li F, et al. Constructing biomedical knowledge graph based on SemMedDB and linked open data. In: *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. Institute of Electrical and Electronics Engineers (IEEE); 2018.
39. Mayers M, Li TS, Queralt-Rosinach N, et al. Time-resolved evaluation of compound repositioning predictions on a text-mined knowledge network. BMC Bioinf 2019;20.
40. Morris JH, Soman K, Akbas RE, et al. The scalable precision medicine open knowledge engine (SPOKE): a massive knowledge graph of biomedical information. In: Lu Z, ed. *Bioinformatics*. Oxford: Oxford University Press (OUP); 2023.
41. Wood EC, Glen AK, Kvarfordt LG, et al. RTX-KG2: a system for building a semantically standardized knowledge graph for translational biomedicine. BMC Bioinf 2022;23.
42. Himmelstein DS, Baranzini SE. Heterogeneous network edge prediction: a data integration approach to prioritize disease-associated genes. PLoS Comput Biol 2015;11:e1004259 .
43. Himmelstein D. *Using the neo4j Graph Database for Hetnets*. ThinkLab; 2015.
44. Himmelstein D, Lizee A. *Estimating the Complexity of Hetnet Traversal*. ThinkLab; 2016.
45. Himmelstein D. *Path Exclusion Conditions*. ThinkLab; 2015.
46. Zietz M.~~ Vagelos Report Summer 2017. *Figshare*~~ https://doi.org/10.6084/m9.figshare.5346577 Vagelos Report Summer 2017. *Figshare*. 2017. **Q28: [Copy Editor Freelancer to Author]** Author: Please provide the URL for the following references: 46, 50, and 52. **[Reply by Author]** Added URLs. They appear to have been inserted into the reference after the first author. Looks like it might need your attention to format correctly.
47. GitHub - mmayers12/hetnet_ml: Software to quickly extract features from heterogeneous networks for machine learning. ~~*GitHub.*~~ Mayers M . https://github.com/mmayers12/hetnet_ml. Accessed 2022 October 4. **Q29: [Copy Editor Freelancer to Author]** Author: Please list authors. **[Reply by Author]** Added the author
48. Hanhijärvi S, Garriga GC, Puolamäki K. Randomization techniques for graphs. In: *Proceedings of the 2009 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics; 2009.
49. Himmelstein D. *Assessing the Effectiveness of Our Hetnet Permutations*. ThinkLab; 2016.
50. Zietz M, https://doi.org/10.1101/2023.01.05.522939 Himmelstein DS, Kloster K, et al. The probability of edge existence due to node degree: a baseline for network-based predictions. Biorxiv 2023. **Q30: [Author to Editor]** The URL for reference 50 did not insert properly. This is a preprint that can be identified with the DOI 10.1101/2023.01.05.522939
51. Lizee A, Himmelstein D. *Network Edge Prediction: Estimating the Prior*. ThinkLab; 2016.
52. Himmelstein D, https://doi.org/10.5281/zenodo.1435833 Zietz M, Kloster K, et al. Node connectivity measurements for Hetionet v1.0 metapaths. *Zenodo*. 2018.

53. Korthauer K, Kimes PK, Duvallet C, et al. A practical guide to methods controlling false discoveries in computational biology. Genome Biol 2019;0.

54. Himmelstein DS, 10.1371/journal.pcbi.1007128 Rubinetti V, Slochower DR, et al. Open collaborative writing with Manubot. PLoS Comput Biol 2019;5:e1007128.

55. Himmelstein D, Zietz M, Rubinetti V, et al. greenelab/connectivity-search-manuscript repository: manuscript source code for Hetnet Connectivity Search. *GitHub*. 2023. https://github.com/greenelab/connectivity-search-manuscript. Accessed 2023 April 6.

56. Himmelstein D, Zietz M, Kloster K, et al. greenelab/connectivity-search-analyses repository: hetnet connectivity search research notebooks. *GitHub*. 2023. https://github.com/greenelab/connectivity-search-analyses. Accessed 2023 April 6.

57. Himmelstein D, Hu D, Alquaddoomi F, et al. greenelab/connectivity-search-backend repository: Django backend for hetnet connectivity search. *GitHub*. 2023. https://github.com/greenelab/connectivity-search-backend. Accessed 2023 April 6.

58. Rubinetti V, Himmelstein D, Hu D, et al. greenelab/connectivity-search-frontend repository: frontend source code for Hetnet connectivity search. *GitHub*. 2023. https://github.com/greenelab/connectivity-search-frontend. Accessed 2023 April 6.

59. Himmelstein D, Zietz M, Kloster K, et al. hetio/hetmatpy repository: Python package for matrix storage and operations on hetnets. *GitHub*. 2022. https://github.com/hetio/hetmatpy. Accessed 2023 April 6.

60. Himmelstein D, Heil B, Zietz M, et al. hetio/hetnetpy repository: Hetnets in Python. *GitHub*. 2021. https://github.com/hetio/hetnetpy. Accessed 2023 April 6.

61. Himmelstein D, Lizee A, Alquaddoomi F, et al. hetio/hetionet: data repository containing Hetionet downloads. *GitHub*. 2023. https://github.com/greenelab/connectivity-search-manuscript. Accessed 2023 April 6.

62. Rubinetti V, Himmelstein D, Hu D, et al. hetio/het.io repository: Source code for het.io website. *GitHub*. 2021. https://github.com/hetio/het.io. Accessed 2023 April 6.

63. Daniel HS, Michael Z, Vincent R, et al. Supporting data for "Hetnet Connectivity Search Provides Rapid Insights into How 2 Biomedical Entities Are Related." *GigaScience Database*. 2023. https://doi.org/10.5524/102389.