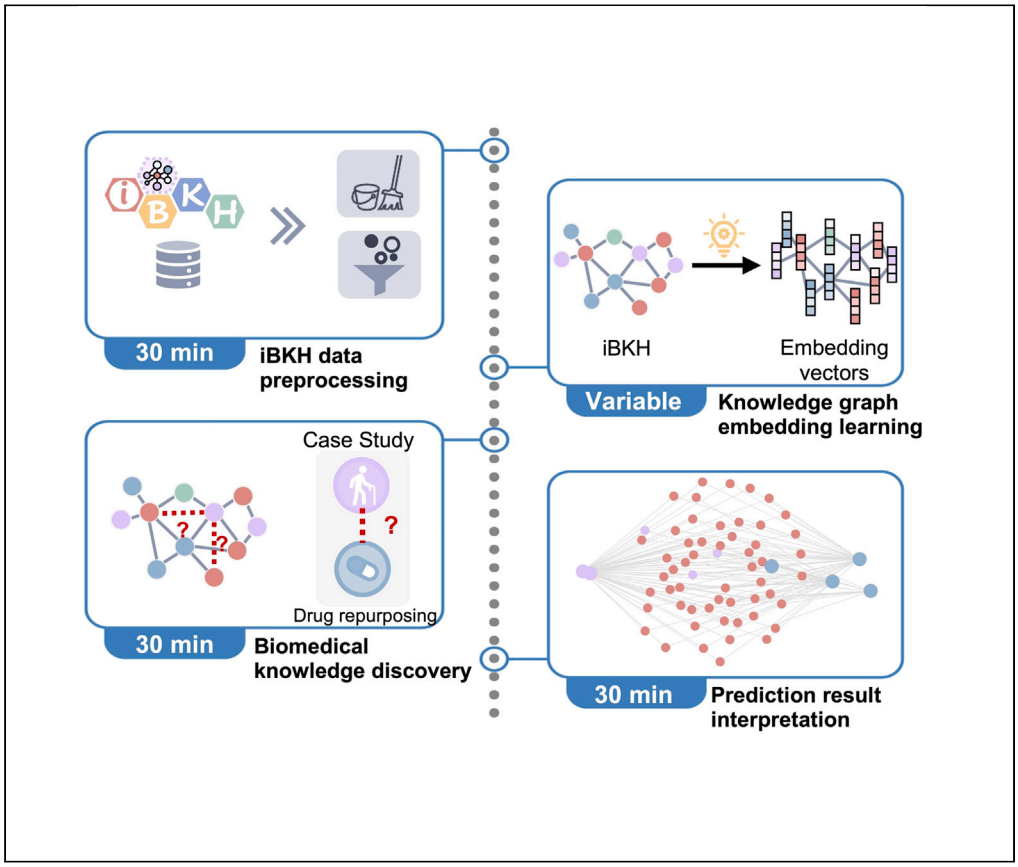
STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol to implement a computational pipeline for biomedical discovery based on a biomedical knowledge graph



Biomedical knowledge graphs (BKGs) provide a new paradigm for managing abundant biomedical knowledge efficiently. Today's artificial intelligence techniques enable mining BKGs to discover new knowledge. Here, we present a protocol for implementing a computational pipeline for biomedical knowledge discovery (BKD) based on a BKG. We describe steps of the pipeline including data processing, implementing BKD based on knowledge graph embeddings, and prediction result interpretation. We detail how our pipeline can be used for drug repurposing hypothesis generation for Parkinson's disease.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Chang Su, Yu Hou, Michael Levin, Rui Zhang, Fei Wang

suchangsc10@gmail.com (C.S.)
few2001@med.cornell.edu (F.W.)

Highlights

A computational pipeline for biomedical knowledge discovery based on graph learning

A case study for *in silico* drug repurposing

Interpreting prediction results based on knowledge graph structure

Su et al., STAR Protocols 4, 102666
December 15, 2023
https://doi.org/10.1016/j.xpro.2023.102666

STAR Protocols

CellPress
OPEN ACCESS

Protocol

Protocol to implement a computational pipeline for biomedical discovery based on a biomedical knowledge graph

Chang Su,[1,4,5,*] Yu Hou,[2,4] Michael Levin,[3] Rui Zhang,[2] and Fei Wang[1,6,*]

[1]Department of Population Health Sciences, Weill Cornell Medicine, New York, NY 10065, USA

[2]Department of Surgery, University of Minnesota, Minneapolis, MN 55455, USA

[3]Bioengineering Department, College of Engineering, Temple University, Philadelphia, PA 19122, USA

[4]These authors contributed equally

[5]Technical contact

[6]Lead contact

*Correspondence: suchangsc10@gmail.com (C.S.), few2001@med.cornell.edu (F.W.)
https://doi.org/10.1016/j.xpro.2023.102666

SUMMARY

Biomedical knowledge graphs (BKGs) provide a new paradigm for managing abundant biomedical knowledge efficiently. Today's artificial intelligence techniques enable mining BKGs to discover new knowledge. Here, we present a protocol for implementing a computational pipeline for biomedical knowledge discovery (BKD) based on a BKG. We describe steps of the pipeline including data processing, implementing BKD based on knowledge graph embeddings, and prediction result interpretation. We detail how our pipeline can be used for drug repurposing hypothesis generation for Parkinson's disease.

For complete details on the use and execution of this protocol, please refer to Su et al.[1]

BEFORE YOU BEGIN

This protocol will give a step-by-step guide to develop a computational pipeline for biomedical knowledge discovery (BKD) based on a comprehensive BKG, the iBKH.[1] We formulate the BKD task in the iBKH as the procedure to predict entities that are potentially linked to the target entity. This protocol includes the following steps including data collection, Python and package installation, knowledge graph embedding, knowledge discovery and evaluation, and prediction result interpretation. We demonstrate a use case of our pipeline for drug repurposing hypothesis generation for Parkinson's disease (PD), i.e., predicting drug entities that could potentially link to the PD entity in the iBKH. This protocol can be also adapted to other BKD tasks such as prediction of disease risk genes[2,3] and drug-drug interaction discovery,[4,5] etc.

Data collection

Timing: 15 min

1. Download the project zip file from GitHub: https://github.com/wcm-wanglab/iBKH/tree/main/iBKH-KD-protocol
2. Unpack the downloaded file.
3. Download the latest version of iBKH knowledge graph (KG) data (entities and relations) at: https://github.com/wcm-wanglab/iBKH/tree/main/iBKH-KD-protocol.

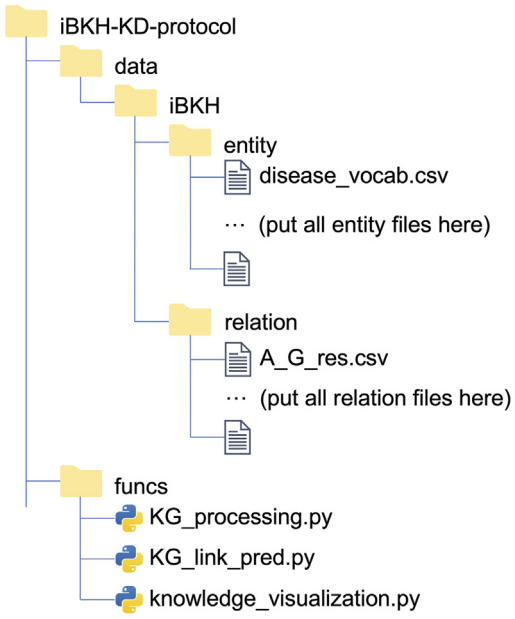


Figure 1. An illustration of file organization of the protocol

4. Put the downloaded files (codes and data) following the structure as shown in Figure 1.

Python and package installation

Timing: 30 min

5. Go to the Anaconda webpage at https://www.anaconda.com/download, download appropriate Anaconda installer according to your computer system, and install it. (Note: please choose Anaconda installer with Python version no later than version 3.7.0)
6. Install required Python packages (NumPy, pandas, scikit-learn, neo4j, networkx) as following:

```
>pip install numpy
>pip install pandas
>pip install sklearn
>pip install neo4j
>pip install networkx
```

7. Install PyTorch.

Note: Installation of PyTorch should follow instructions at: https://pytorch.org.

a. Navigate to the "PyTorch Build" section, and there, choose the "Stable" version from the available choices.
b. Move to the "Your OS" section and choose the appropriate operating system you're using.
c. Decide on the installation approach you intend to employ (we recommend either "pip" or "conda").
d. Indicate that you are working with Python as your chosen programming language for PyTorch.

Note: For this procedure, we don't need a GPU, so proceed to select the "Default" option within the "Compute Platform" category.

e. Once these selections are made, copy the automatically generated command under the "Run this Command" section and execute it in the command line (for Windows and Linux users) or in the terminal (for Mac OS users).

Note: This will initiate the installation process.

8. Install the DGL-KE (Deep Graph Library – Knowledge Embedding) package.[6] DGL-KE is a Python-based implementation for the deep learning (DL)-based knowledge graph embedding algorithms. Installation of DGL-KE follows instructions at: https://github.com/awslabs/dgl-ke. Specifically, run the following commands to initiate the installation process for DGL-KE:

```
>sudo pip install dgl

>sudo pip install dgl-ke
```

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Anaconda (with Python 3.7.0 or later version)	Anaconda Inc.	RRID: SCR_018317; https://www.anaconda.com/download
iBKH	Our recent paper[1]	http://ibkh.ai and https://github.com/wcm-wanglab/iBKH
Source code	This paper	https://github.com/wcm-wanglab/iBKH/tree/main/iBKH-KD-protocol https://doi.org/10.5281/zenodo.8371233
Software and algorithms		
NumPy 1.21.5 or later version	Harris et al.[7]	SCR: 008633; https://numpy.org/
pandas 1.3.5 or later version	NumFOCUS Inc.	https://pandas.pydata.org
neo4j 5.3.0 or later version	Neo4j Inc.	https://neo4j.com/developer/python/
Network 2.6.3 or later version	NetworkX Developers	https://networkx.org
scikit-learn 0.23.1 or later version	Buitinck et al.[8]	https://scikit-learn.org/stable/
PyTorch 1.4.0 or later version	PyTorch Foundation	https://pytorch.org
DGL-KE	Zheng et al.[6]	https://github.com/awslabs/dgl-ke

STEP-BY-STEP METHOD DETAILS

The following are detailed instructions on how to how to implement the biomedical knowledge discovery pipeline. We show examples of each step in a tutorial Jupyter Notebook project called "Knowledge_Discovery_Pipeline.ipynb", which can be found in our GitHub repository.

iBKH data preprocessing

Timing: 30 min

Note: This section introduces steps for preprocessing the iBKH BKG data.

This protocol uses a comprehensive BKG we built, termed iBKH.[1] Figure 2 illustrates the schema of iBKH. Currently, iBKH includes 11 entity types (including anatomy, disease, drug, gene, molecule,

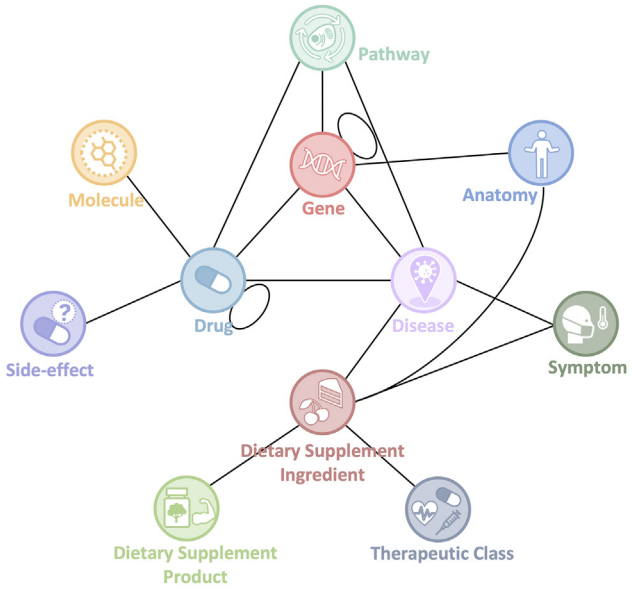


Figure 2. Schema of iBKH knowledge graph

Each circle denotes an entity type, and each link denotes a meta relation between a pair of entities. Of note, a meta relation can represent multiple types of relations between a specific pair of entities. For example, five potential relations including 'Associates', 'Downregulates', 'Upregulates', 'Inferred_Relation', 'Text_Semantic_Relation' can exist between a pair of disease and gene entities.

symptom, pathway, side effect, dietary supplement ingredient [DSI], dietary supplement product [DSP], and dietary's therapeutic class [TC]) and 45 relation types within different entity pairs such as Drug- Disease (DDi), Drug-Drug (DD), Drug-Gene (DG), etc.

In a BKG, like the iBKH, a triplet is the smallest unit for storing information. Typically, a triplet can be formulated as (h,r,t), where h and t are the head and tail entities, and r is the relation linking h to t. This section describes the steps for iBKH KG data preprocessing, i.e., extracting and formatting triplets from the iBKH, which will be used to train knowledge graph embedding models.

1. Open the Anaconda-Navigator and launch the Jupyter Notebook.
2. In the Jupyter Notebook interface, run the following codes to import required packages.

```
>import pandas as pd

>import numpy as np

>import pickle

>import torch as th

>import torch.nn.functional as fn

>import os

>import sys

>sys.path.append('.')

>import funcs.KG_processing as KG_processing
```

3. Extract triplets from raw data of iBKH.
   a. Set up the input and output file paths.

```
> /* Input iBKH-KG data path */
>kg_folder = 'Data/iBKH/'
> /* Output path */
>triplet_path = 'Data/triplets/'
>if not os.path.exists(triplet_path):
> os.makedirs(triplet_path)
> /* Output data file path */
>output_path = 'Data/dataset/'
>if not os.path.exists(output_path):
> os.makedirs(output_path)
```

   b. Extract triplets of different entity pair types by running following codes.

```
>KG_processing.DDi_triplets(kg_folder, triplet_path)
>KG_processing.DG_triplets(kg_folder, triplet_path)
>KG_processing.DPwy_triplets(kg_folder, triplet_path)
>KG_processing.DSE_triplets(kg_folder, triplet_path)
>KG_processing.DiDi_triplets(kg_folder, triplet_path)
>KG_processing.DiG_triplets(kg_folder, triplet_path)
>KG_processing.DiPwy_triplets(kg_folder, triplet_path)
>KG_processing.DiSy_triplets(kg_folder, triplet_path)
>KG_processing.GG_triplets(kg_folder, triplet_path)
>KG_processing.GPwy_triplets(kg_folder, triplet_path)
>KG_processing.DD_triplets(kg_folder, triplet_path)
```

   Note: This will result in a set of CSV files in the "iBKH-KD-protocol/data/triplets/", storing triplets regarding each entity pair type.

   c. Combine the triplets to generate a TSV file based on the DGL-KE input requirement.

```
> /* Specify triplet types you want to use. */
>included_pair_type = ['DDi', 'DG', 'DPwy', 'DSE', 'DiDi', 'DiG',
'DiPwy', 'DiSy', 'GG', 'GPwy', 'DD']
> /* Combine triplets */
>KG_processing.generate_triplet_set(triplet_path=triplet_path)
> /* Generate DGL-KE required input triplet file */
```

```
>KG_processing.generate_DGL_training_set(triplet_path=triplet_path,\
output_path=output_path)
```

Note: The variable "included_pair_type" specifies a list of triplet types that we plan to use for analysis. The generated data files can be found in the folder "iBKH-KD-protocol/data/dataset/", including "training_triplets.txt", "validation_triplets.tsv", and "testing_triplets.tsv", which will be used for training and evaluating the knowledge graph embedding models, as well as "whole_triplets.tsv", which will be used for training the final models.

Knowledge Graph Embedding Learning

Timing: variable depending on hardware, approximately 8–24 h

Note: This section introduces steps for learning embedding vectors for entities and relations in the iBKH.

Knowledge graph embedding aims to learn machine-readable embedding vectors for entities and relations in a BKH (e.g., the iBKH) while preserving the graph structure.[9,10] We engage four deep learning-based knowledge graph embedding algorithms implemented in the DGL-KE, including TransE,[11] TransR,[12] ComplEx,[13] and DistMult.[14] This section describes the steps for training the models.

4. This step trains each knowledge graph embedding model (TransE, TransR, ComplEx, and DistMult) using the iBKH.
   a. Open command line (Windows OS and UNIX OS) or terminal (MAC OS) and change directory to the project as below.

```
>cd [your file path]/iBKH-KD-protocol
```

   b. Train and evaluate the knowledge graph embedding model using below command:

```
> DGLBACKEND=pytorch \
dglke_train --dataset iBKH --data_path ./data/dataset \
--data_files training_triplets.tsv \
validation_triplets.tsv \
testing_triplets.tsv \
--format raw_udd_hrt --model_name [model name] \
--batch_size [batch size] --hidden_dim [hidden dim] \
--neg_sample_size [neg sample size] --gamma [gamma] \
--lr [learning rate] --max_step [max step] \
--log_interval [log interval] \
--batch_size_eval [batch size eval] \
-adv --regularization_coef [regularization coef] \
--num_thread [num thread] --num_proc [num proc] \
```

```
--neg_sample_size_eval [neg sample size eval] \
--save_path ./data/embeddings --test
```

Note: We use multiple measurements to evaluate model performances including: HITS@k, the average number of times the positive triplet is among the k highest ranked triplets; Mean Rank (MR), the average rank of the positive triplets; and Mean Reciprocal Rank (MRR), the average reciprocal rank of the positive instances. Higher values of HITS@k and MRR and a lower value of MR indicate good performance, and vice versa. Some useful arguments of the DGL-KE command are listed in Table 1. Detailed instructions for the DGL-KE commands can be found at: https://dglke.dgl.ai/doc/train.html.

c. Once the model can achieve a desirable performance in the testing set, we can re-train the model using the whole dataset by running:

```
> DGLBACKEND=pytorch \
dglke_train --dataset iBKH --data_path ./data/dataset \
--data_files whole_triplets.tsv \
--format raw_udd_hrt --model_name [model name] \
--batch_size [batch size] --hidden_dim [hidden dim] \
--neg_sample_size [neg sample size] --gamma [gamma] \
--lr [learning rate] --max_step [max step] \
--log_interval [log interval] \
-adv --regularization_coef [regularization coef] \
--num_thread [num thread] --num_proc [num proc] \
--save_path ./data/embeddings
```

Note: This will generate two output files for each model: "iBKH_[model name]_entity.npy", containing the low dimension embeddings of entities in iBKH and "iBKH_[model name]_relation.npy", containing the low dimension embeddings of relations in iBKH. These embeddings can be used in downstream BKD tasks.

We run above procedures based on TransE, TransR, ComplEx, and DistMult, respectively, to gain embedding vectors of entities and relations in the iBKH.

Note: The user may repeat the Step 4b multiple times to find the optimal hyperparameters of each model. Here, we share the optimal hyperparameter values we found in our experiments as listed in Table 2. For simplicity, the user can directly use the suggested hyperparameter values to train the models. In addition, running time of the knowledge graph embedding procedure varies, depending on hardware used. For our experiments, we used a machine equipped with an Intel i7-7800X CPU, boasting 6 cores and 12 threads, with a fundamental clock speed of 3.5 GHz, coupled with 62 GB of RAM. Training the four knowledge graph embedding models within the dataset took approximately 8 hours in our experiment. The required running time could extend to 24 hours or even more if a user expects to tune the models to find the optimal hyperparameters for enhancing model performance.

Table 1. Some useful arguments for training a knowledge graph embedding model using DGL-KE

Arguments	Description
–dataset	Dataset name.
–data_path	Folder where the dataset is stored.
–data_files	A list of input data files including [training data file], [validation data file], and [testing data file], each of which contains a list of triplets extracted from the BKG. The [training data file] is required and used to train the model, while [validation data file] and [testing data file] are optional and used for evaluating model performance.
–model_name	Model used. DGL-KE provides multiple knowledge graph embedding models. In the protocol, we suggest using TransE_l2, TransR, DistMult, and ComplEx, whose robustness in iBKH has been validated in our previous work.[1]
–batch_size	The batch size for training.
–batch_size_eval	The batch size used for model evaluation.
–neg_sample_size	The number of negative samples we use for each positive sample in the training.
–neg_sample_size_eval	The number of negative samples we use to evaluate a positive sample.
–hidden_dim	The sizes of the learned embeddings of relations and entities.
–gamma	The margin value in the score function. It is used by TransE and TransR.
–lr	The learning rate.
–max_step	The maximal number of steps to train the model in a single process.
–log_interval	Print runtime of different components every log_interval steps.
–regularization_coef	The coefficient for regularization.
–num_thread	The number of CPU threads to train the model in each process.
–num_proc	The number of processes to train the model in parallel.
–test/–valid	Evaluate the model on the testing/validation set after the model is trained.
–save_path	The folder where embedding vectors and logs are saved.
More details can be found at: https://dglke.dgl.ai/doc/train.html	

Biomedical knowledge discovery – biomedical hypothesis generation

Timing: 30 min

Note: This section introduces the implementation of BKD based on knowledge graph embeddings learned from iBKH.

Here, we showcase a case study of drug repurposing hypothesis generation for Parkinson's disease (PD).

5. Turn to the Jupyter Notebook interface and run the following script to import required package packages.

```
>from funcs.KG_link_pred import generate_hypothesis,\

generate_hypothesis_ensemble_model
```

6. Define the PD entity using

```
> PD = ["parkinson's disease", "late onset parkinson's disease"]
```

Table 2. Suggested hyperparameters for training the knowledge graph embedding models

Arguments	TransE	TransR	ComplEx	DistMult
–model_name	TransE_l2	TransR	ComplEx	DistMult
–batch_size	1024	1024	1024	1024
–batch_size_eval	1000	1000	1000	1000
–neg_sample_size	256	256	256	256
–neg_sample_size_eval	1000	1000	1000	1000
–hidden_dim	400	200	200	400
–gamma	12.0	12.0	12.0	12.0
–lr	0.1	0.005	0.005	0.005
–max_step	10000	10000	10000	10000
–log_interval	100	100	100	100
–regularization_coef	1.00E-09	1.00E-07	1.00E-07	1.00E-07

Note: Here we collect a list of PD terms. These PD terms can be obtained in the entity vocabularies in the "data/iBKH/entity" folder.

7. The task is to predict drug entities that don't have "treats" and "palliates" relationships with PD in the iBKH but can potentially treat or palliate PD. Therefore, we define a relation type list:

```
> r_type = ["Treats_DDi", "Palliates_DDi"]
```

Note: More relation types can be found in the "data/iBKH/relation" folder.

8. Predict repurposable drug candidates for PD (in this example, we use embedding vectors based on the TransE model for prediction):

```
> proposed_df = generate_hypothesis(target_entity=PD,
candidate_entity_type='drug',
relation_type=r_type,
embedding_folder='data/embeddings',
method='transE_l2',
kg_folder = 'data/iBKH',
triplet_folder = 'data/triplets',
topK=100, save_path='output',
save=True, without_any_rel=False)
```

Running the above code will result in an output CSV file within the "output" folder, which stores top-100 ranked repurposable drug candidates for PD based on the TransE model.

Note: Please refer to Table 3 for detailed information regarding arguments of the function.

9. Using the code in Step 8 can make predictions based on a single knowledge graph embedding model (TransE in the example). To enhance prediction performance, we also proposed an ensemble model, which combines the four embedding algorithms to make predictions. In our

preliminary work, we have demonstrated that the ensemble model can improve knowledge discovery performance in iBKH.[1] The following code introduces the usage of the ensemble model to predict repurposable drug candidates for PD.

```
> proposed_df = generate_hypothesis_ensemble_model (target_entity=PD,
candidate_entity_type='drug',
relation_type=r_type,
embedding_folder='data/embeddings',
kg_folder = 'data/iBKH',
triplet_folder = 'data/triplets',
topK=100, save_path='output',
save=True, without_any_rel=False)
```

Running the above code will result in an output CSV file within the "output" folder, which stores top-100 ranked repurposable drug candidates for PD based on the ensemble model.

Note: Please refer to Table 3 for detailed information regarding arguments of the function.

Prediction result interpretation

Timing: 30 min

Note: This section introduces the procedure of interpreting the prediction results based on the iBKH.

We extract the shortest paths that connect the target entity (e.g., PD) with the predicted entities (e.g., the predicted repurposing drug candidates of PD) to generate a contextual subnetwork.

10. Taking the PD drug repurposing task as an example, we can generate the contextual subnetwork surrounding PD and some predicted repurposing drug candidates as below.
    a. Import required package.

```
>from funcs.knowledge_visualization as kv
```

    b. Specify the predicted drug candidates to interpret. Here, we focus on the top four candidates predicted using the ensemble model, including Glutathione, Clioquinol, Steroids, and Taurine.

```
> drug_list = ['Glutathione', 'Clioquinol', 'Steroids', 'Taurine']
```

    c. Create a contextual subnetwork linking PD and the drug candidates.

```
> kv.subgraph_visualization(target_type='Disease',
target_list=PD,
```

```
predicted_type='Drug',

predicted_list=drug_list,

neo4j_url = "neo4j://54.210.251.104:7687",

username = "neo4j", password = "password",

alpha=1.5, k=0.8, figsize=(15, 10),

save=True)
```

This will result in a figure saved as a PDF file in the "output" folder. Please refer to Table 4 for detailed information regarding arguments of the function.

Note: The shortest path query is based on iBKH deployed using the Neo4j, an efficient graph database. Please refer to https://github.com/wcm-wanglab/iBKH#neo4j-deployment for detailed information if you want to create your own iBKH Neo4j instance.

EXPECTED OUTCOMES

The expected outcome of this protocol is the predicted new knowledge.

Following instructions in Steps 1–4, we could extract iBKH knowledge graph data and conduct knowledge graph embedding with different algorithms including TransE, TransR, complEx, and DistMult. First, each model will be trained in the training set and evaluated in the testing set (see Step 3). This will result in link prediction performance of the embedding models. Of note, we can repeat Step 4 multiple times to find the optimal model hyperparameters (see Table 1) to enhance the learned embeddings. For convenience, we suggest the optimal hyperparameter values we found as listed in Table 2. Given this, we could re-train the models using the whole data set, which will result in entity and relation embedding vectors saved in the .npy files.

Table 3. Descriptions of arguments for hypothesis generation (the generate_hypothesis and generate_hypothesis_ensemble_model functions)

Arguments	Description
target_entity	A list of terms of the target entity. Please refer to entity vocabularies under the "data/iBKH/entity" directory. Note: only the names of the entities should be used, and it is case sensitive.
candidate_entity_type	A list of relation types to predict, linking the target entities and candidate entities. Note: details of relation types can be found under the "data/iBKH/relation" directory.
embedding_folder	The folder that saves knowledge graph embedding results obtained in the previous step. Default: "data/embeddings".
method	Knowledge graph embedding method used. Options: transE_l2, transR, DistMult, and ComplEx. Default: transE_l2. Note: only applicable to the function: generate_hypothesis.
kg_folder	The folder that saves the raw knowledge graph data. Default: "data/iBKH".
triplet_folder	The batch size used for validation and test.
without_any_rel	If True, exclude candidate entities that have any relations with the target entity. If False, only exclude candidate entities that have already linked to the target entity with relations in the candidate_entity_type. Default: False.
topK	The output result will include the top K ranked candidate entities that potentially link to the target entity. Default: 100.
save_path	The folder to save the output results. Default: "output".
save	This parameter determines whether the output results are saved in the designated save_path. If True, the results will be stored; if False, the results will not be saved. Default: True.

Table 4. Descriptions of arguments for prediction result interpretation

Arguments	Description
target_type	Type of the target entity.
target_list	A list of terms of the target entity. Please refer to entity vocabularies under the "data/iBKH/entity" directory. Note: only the names of the entities should be used, and it is case sensitive.
predicted_type	Type of the predicted entities.
predicted_list	A list of predicted entities.
excluded_r_type	A list of relation types to be excluded for visualization. Default: [].
neo4j_url	URL of Neo4j knowledge graph instance.
user_name	Username of Neo4j instance.
password	Password of Neo4j instance.
alpha	Float. Alpha controls space between the target and predicted entities in the illustration. Default: 1.
K	Float. It controls distance between nodes. Default: 0.3.
nsize	Node size.
target_size_ratio	The ratio of target/predicted nodes to intermediate nodes. 1, all nodes have the same size; >1, amplifying the target/predicted nodes. Default: 2.5.
with_node_label	Show node label if True. Default: True.
node_label_size	Node label font size. Default: 10.
with_edge_label	Show edge label if True. Default: True.
edge_label_size	Edge label font size. Default: 7.
figsize	Tuple. Figure size. Default: (14, 10).
save	Save output figure as a file if True.
save_path	The folder to save the output figure. Default: "output".
save_name	The name of the output file. Default: None.

Following instructions in Steps 5–9, we could conduct knowledge discovery. Taking the PD drug repurposing hypothesis generation task as an example, we can obtain a list of top-ranked potential drug candidates as shown in Figure 3.

Additionally, following instructions in Step 10, we can extract the shortest paths linking PD and predicted drug candidates of interest, in the iBKH. This will result in a subnetwork as illustrated in Figure 4.

LIMITATIONS

First, we used our iBKH knowledge graph in this protocol. iBKH has integrated data from a wide spectrum of sources; however, the information contained therein can still be incomplete due to the volume and speed of the new biomedical knowledge that has been generated everyday.[1] For instance, knowledge regarding entities like protein, protein structure, complex, mutation, etc. are important resources but haven't been included into iBKH yet. This may limit the capacity of our approach in discovering knowledge related to these entities. . To address this, we will make curating and adding new information into iBKH a continuous effort to enhance biomedical knowledge discovery.

Second, this protocol leveraged knowledge graph embedding algorithms including TransE, TransR, complEx, and DistMult which have demonstrated state-of-the-art performances. Beyond these, other models like graph neural networks have also demonstrated significant performance in knowledge discovery based on knowledge graph.[15,16] We will incorporate more models to advance our pipeline in the future.

Third, it is important to validate the novel knowledge discovered from iBKH. The current pipeline supports generating a subnetwork based on shortest paths linking the target and predicted entities

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	primary	name	drugbank_id	kegg_id	pharmgkb_id	umls_cui	mesh_id	iDISK_id	CID	id	score	score_norm	transE_vote	transR_vote	complEx_vote	DistMult_vot	vote_score	vote_score_normed
2	DrugBank:DB00143	Glutathione	DB00143	D00014	PA449780	C0017817	D005978			3092	-1.3192606	0.81111217	24263	26927	26138	24317	101645	1
3	DrugBank:DB04815	Clioquinol	DB04815	D03538	PA449039	C0021978	D007464			1603	-0.9699813	0.86587286	24994	26876	24297	25295	101462	0.998049934
4	MeSH:D013256	Steroids				C0038317	D013256			14378	-1.3089085	0.8127352	24275	25877	24796	26138	101086	0.994043242
5	DrugBank:DB01956	Taurine	DB01956	D00047	PA451590	C0039350	D013654			3451	-1.1711981	0.8343257	24518	26020	25620	24751	100909	0.992157113
6	DrugBank:DB00431	Lindane	DB00431	D00360	PA164754914	C0005038	D001556		CID10000072	1308	-0.6793266	0.9114423	25776	26705	25516	22897	100894	0.991997272
7	DrugBank:DB00132	alpha-Linolenic acid	DB00132		PA449384	C0051405	D017962			6868	-1.6717641	0.7558459	23680	26141	25097	25957	100875	0.991794806
8	DrugBank:DB00179	Masoprocol	DB00179	D04862	PA164746493	C0733397	D009637			11706	-1.6596137	0.75775087	23697	26983	24552	25515	100747	0.990430826
9	DrugBank:DB00336	Nitrofural	DB00336	D00862	PA164754877	C0028157	D009583			12524	-1.3049827	0.8133507	24280	25621	25051	25766	100718	0.990121799
10	DrugBank:DB04115	Berberine	DB04115	D00092	PA165860812	C0005117	D001599			7823	-1.3932794	0.7995073	24156	25945	25326	25269	100696	0.989887365
11	DrugBank:DB09086	Eugenol	DB09086	D04117		C0015153	D005054			10173	-1.7526864	0.7431588	23552	26093	24917	26097	100659	0.98949309
12	DrugBank:DB12290	Puerarin	DB12290			C0072591	C033607			13634	-1.7054859	0.750559	23620	26102	24885	25981	100588	0.988736507
13	MeSH:D008094	Lithium				C0023870	D008094	DC0478809	CID10001112	11539	-0.5504055	0.9316548	26175	26771	25921	21578	100445	0.987212685
14	DrugBank:DB11118	Ammonia	DB11118		PA166131585	C0002607	D000641		CID10000022	7068	-1.2208875	0.8265353	24419	26092	24718	25118	100347	0.986168388
15	DrugBank:DB02587	Colforsin	DB02587	D03584	PA146096022	C0917964	D005576			8994	-1.310175	0.8125366	24272	26546	25592	23826	100236	0.984985561
16	DrugBank:DB11735	Galactose	DB11735	D04291	PA449725	C0016945	D005690			10530	-1.7624575	0.74162686	23538	26153	23936	26573	100200	0.984601942
17	PharmGKB:PA10832	corticosteroids			PA10832	C0001617	D000305	DC0478594		6611	-1.385069	0.8007946	24169	25704	24801	25497	100171	0.984292915
18	DrugBank:DB04422	Homocysteine	DB04422			C0019878	D006710			10902	-1.2244079	0.82598335	24414	26287	24528	24890	100119	0.983738798
19	DrugBank:DB09153	Sodium chloride	DB09153	D02056	PA451382	C0037494	D012965			1522	-1.5519575	0.7746295	23890	26184	25322	24645	100041	0.982907622
20	DrugBank:DB11672	Curcumin	DB11672		PA151958596	C0010467	D003474			9140	-1.250675	0.82186514	24354	27049	26388	22138	99929	0.98171414
21	MeSH:D002331	Carnitine				C0007258	D002331		CID10000008	8437	-1.4813418	0.78570074	24020	26586	26241	22912	99759	0.979902603

Figure 3. An illustration of predicted potential drug candidates (top 20) for Parkinson's disease using our protocol

to interpret the results. As a relevant effort, we have built a biomedical evidence generation engine based on literature mining, which can retrieve and synthesize evidence supporting particular hypotheses from state-of-the-art scientific publications.[17] Taking the drug repurposing task as an example, for each predicted drug candidate, we can retrieve relevant literature that discussed the drug and the target disease. After that, we can generate a piece of textual summary to demonstrate early evidence regarding potential relationships of the two entities. We plan to add this new functionality to our pipeline in the future.

TROUBLESHOOTING

Problem 1

Fail to install the required Python packages (before you begin – python and packages installation).

Potential solution

Create a Conda virtual environment and install the packages following our instruction. Detailed information for creating a Conda virtual environment can be found at: https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html.

Problem 2

iBKH currently contains over 2 million entities and 50 million relations among them. Training the knowledge graph embedding algorithms in such a huge graph data is time-consuming and the running time highly depends on hardware used. (step-by-step method details – knowledge graph embedding learning).

Potential solution

To address this issue to obtain the embedding vectors of entities and relations efficiently, potential solutions include.

- Train the knowledge graph embedding models using a high-performance computation platform equipped with more CPU cores ($\geq$ 4) with high processing speed and memory $\geq$ 16GB.
- Use less triplet types. For instance, in the drug repurposing task, to save computation resource, we can use included_pair_type = ['DDi', 'DiG', 'DG', 'GG', 'DD', 'DiDi'] (see Step 3 in the instruction).
- Use our suggested hyperparameter values to train the models (Table 2).
- Use our pre-trained embeddings. Specifically, download the pre-trained embedding vectors at https://drive.google.com/drive/folders/1HYDepbB2Vb2fMaHl_WjPwhWikRNJ5Lq5?usp=share_link, and put the entire folder into the "./iBKH-KD-protocol/data" directory. Then, the user can move to Step 5 for knowledge discovery.

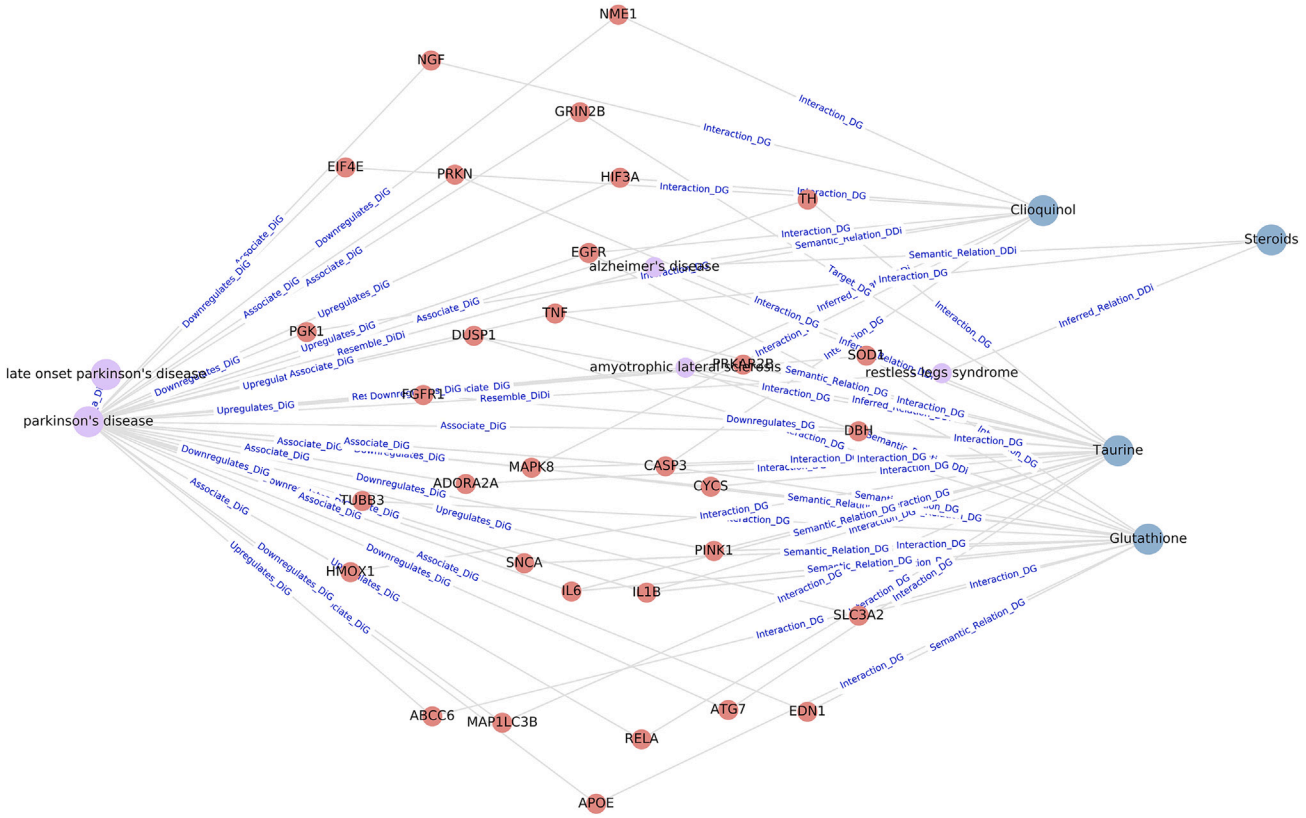


Figure 4. An illustration of subnetwork indicating the shortest paths that link Parkinson's disease and four selected repurposable drug candidates (Glutathione, Clioquinol, Steroids, and Taurine) in the iBKH

Problem 3

In knowledge graph embedding model fine-tuning, operating the DGL-KE command in Step 4 in a large hyperparameter searching space is difficult. (step-by-step method details – knowledge graph embedding learning).

Potential solution

We can create a terminal-executable Shell script to conduct grid search in the hyperparameter space. For instance, we can create a Shell script named "fine-tune.sh" as below.

```
>#!/bin/bash
>seq_num=0
>for embed_size in 200 400
>do
>for lr in 0.001 0.005 0.01 0.05 0.1
>do
>echo "$embed_size; $lr"
>DGLBACKEND=pytorch \
dglke_train --dataset iBKH --data_path ./data/dataset \
```

```
--data_files whole_triplets.tsv --format raw_udd_hrt \

--model_name [model name] --batch_size [batch size] \

--hidden_dim $embed_size --gamma [gamma]

--lr $lr --max_step [max step]

--log_interval [log interval] -adv

--regularization_coef [regularization coef]

--num_thread [num thread] --num_proc [num proc]

>done

>done

Then, turn to the command line (Windows OS and UNIX OS) or terminal (MAC OS), and run the commands
below

>sudo chmod 777 fine-tune.sh

>sh fine-tune.sh
```

Problem 4

Entities (nodes) and relations (edges) appear disorganized and improperly configured in the subnetwork visualization (step-by-step method details – prediction result interpretation).

Potential solution

Increase alpha, k, and target_size_ratio, and decrease nsize in the subgraph_visualization function (Table 4).

Problem 5

Fail to generate subnetwork for prediction result interpretation due to failure of access to iBKH instance in AWS. (step-by-step method details – prediction result interpretation).

Potential solution

Create a new iBKH instance using an Amazon Web Services server (AWS) or local server. Detailed instructions can be found at: https://github.com/wcm-wanglab/iBKH#neo4j-deployment. Modify the Neo4j login information ("neo4j_url", "user_name", and "password") accordingly to generate and visualize the subnetwork for interpreting results.

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Prof. Fei Wang (few2001@med.cornell.edu).

Materials availability

This study did not generate any reagents.

Data and code availability

- The harmonized entity and relation source files for iBKH knowledge graph in CSV (comma-separated values) format are publicly available online at https://github.com/wcm-wanglab/iBKH.
- The computer codes are publicly available online at https://github.com/wcm-wanglab/iBKH/tree/main/iBKH-KD-protocol.
- Any additional information required to reanalyze the data reported in this paper is available from the lead contact upon request.

ACKNOWLEDGMENTS

The authors would like to acknowledge the support from National Science Foundation (NSF; 1750326 and 2212175) and National Institutes of Health (NIH; R01AG076234, R01AG076448, RF1AG072449, R01AG080991, R01AG080624, R01AG078154, and R01AT009457) for this research.

AUTHOR CONTRIBUTIONS

Conceptualization, F.W. and C.S.; writing, C.S. and Y.H.; review and editing, F.W., C.S., Y.H., M.L., and R.Z.; funding acquisition, F.W. and R.Z.; supervision, F.W. and C.S.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

1. Su, C., Hou, Y., Zhou, M., Rajendran, S., Maasch, J.R.M.A., Abedi, Z., Zhang, H., Bai, Z., Cuturrufo, A., Guo, W., et al. (2023). Biomedical discovery through the integrative biomedical knowledge hub (iBKH). iScience *26*, 106460. https://doi.org/10.1016/j.isci.2023.106460.

2. Peng, C., Dieck, S., Schmid, A., Ahmad, A., Knaus, A., Wenzel, M., Mehnert, L., Zirn, B., Haack, T., Ossowski, S., et al. (2021). CADA: phenotype-driven gene prioritization based on a case-enriched knowledge graph. NAR Genom. Bioinform. *3*, lqab078. https://doi.org/10.1093/nargab/lqab078.

3. Hu, J., Lepore, R., Dobson, R.J.B., Al-Chalabi, A., M Bean, D., and Iacoangeli, A. (2021). DGLinker: flexible knowledge-graph prediction of disease–gene associations. Nucleic Acids Res. *49*, W153–W161. https://doi.org/10.1093/nar/gkab449.

4. Celebi, R., Uyar, H., Yasar, E., Gumus, O., Dikenelli, O., and Dumontier, M. (2019). Evaluation of knowledge graph embedding approaches for drug-drug interaction prediction in realistic settings. BMC Bioinf. *20*, 726. https://doi.org/10.1186/s12859-019-3284-5.

5. Ren, Z.-H., Yu, C.-Q., Li, L.-P., You, Z.-H., Guan, Y.-J., Wang, X.-F., and Pan, J. (2022). BioDKG–DDI: predicting drug–drug interactions based on drug knowledge graph fusing biochemical information. Brief. Funct. Genomics *21*, 216–229. https://doi.org/10.1093/bfgp/elac004.

6. . Article in a book Zheng, D., Song, X., Ma, C., Tan, Z., Ye, Z., Dong, J., Xiong, H., Zhang, Z., and Karypis, G. (2020). DGL-KE: Training knowledge graph embeddings at scale. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (Association for Computing Machinery), pp. 739–748. https://doi.org/10.1145/3397271.3401172.

7. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., et al. (2020). Array programming with NumPy. Nature *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2.

8. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., and Grobler, J. (2013). API design for machine learning software: experiences from the scikit-learn project. Preprint at arXiv. https://doi.org/10.48550/arXiv.1309.0238.

9. Su, C., Tong, J., Zhu, Y., Cui, P., and Wang, F. (2020). Network embedding in biomedical data science. Brief. Bioinform. *21*, 182–197. https://doi.org/10.1093/bib/bby117.

10. Mohamed, S.K., Nounu, A., and Nováček, V. (2021). Biological applications of knowledge graph embedding models. Brief. Bioinform. *22*, 1679–1693. https://doi.org/10.1093/bib/bbaa012.

11. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating Embeddings for Modeling Multi-Relational Data. Advances in Neural Information Processing Systems (NIPS 2013). https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html.

12. Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. (2015). Learning Entity and Relation Embeddings for Knowledge Graph Completion. Twenty-Ninth AAAI Conference on Artificial Intelligence. https://ojs.aaai.org/index.php/AAAI/article/view/9491.

13. Théo, T., Johannes, W., Sebastian, R., Eric, G., and Guillaume, B. (2016). Complex Embeddings for Simple Link Prediction. Proceedings of the 33rd International Conference on Machine Learning, Held in New York, USA, 2016/06/11. (JMLR.Org). http://proceedings.mlr.press/v48/trouillon16.html?ref=https://githubhelp.com. 2071–2080.

14. Yang, B., Yih, S.W.-t., He, X., Gao, J., and Deng, L. (2015). Embedding Entities and Relations for Learning and Inference in Knowledge Bases. International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1412.6575.

15. Arora, S. (2020). A survey on graph neural networks for knowledge graph completion. Preprint at arXiv. https://doi.org/10.48550/arXiv.2007.12374.

16. Ye, Z., Kumar, Y.J., Sing, G.O., Song, F., and Wang, J. (2022). A Comprehensive Survey of Graph Neural Networks for Knowledge Graphs. IEEE Access *10*, 75729–75741. https://doi.org/10.1109/ACCESS.2022.3191784.

17. Zhao, S., Wang, A., Qin, B., and Wang, F. (2022). Biomedical evidence engineering for data-driven discovery. Bioinformatics *38*, 5270–5278. https://doi.org/10.1093/bioinformatics/btac675.