Supporting Information

A Workflow of Integrated Resources to Catalyze Network-Pharmacology Driven COVID-19 Research

Gergely Zahoránszky-Kőhalmi [*,1], Vishal B. Siramshetty[1], Praveen Kumar[2,3], Manideep Gurumurthy[1], Busola Grillo[1], Biju Mathew[1], Dimitrios Metaxatos[1], Mark Backus[1], Tim Mierzwa[1], Reid Simon[1], Ivan Grishagin[1,4], Laura Brovold[4], Ewy A. Mathé[1], Matthew D. Hall[1], Samuel G. Michael[1], Alexander G. Godfrey[1], Jordi Mestres[5], Lars J. Jensen[6], Tudor I. Oprea[*,2, 6, 7, 8]

[1]National Center for Advancing Translational Sciences, Rockville, 9800 Medical Center Dr., MD 20850, USA

[2]Department of Internal Medicine, University of New Mexico School of Medicine, 1 University of New Mexico, Albuquerque, NM 87131, USA

[3]Department of Computer Science, University of New Mexico, 1 University of New Mexico Albuquerque, NM 87131, USA

[4]Rancho BioSciences LLC., 16955 Via Del Campo Suite 200, San Diego, CA 92127, USA

[5]Research Group on Systems Pharmacology, Research Program on Biomedical Informatics (GRIB), IMIM Hospital del Mar Medical Research Institute and University Pompeu Fabra, Doctor Aiguader 88, 08003 Barcelona, Catalonia, Spain.

[6]Novo Nordisk Foundation Center for Protein Research, Faculty of Health and Medical Sciences, University of Copenhagen, Blegdamsvej 3B, 2200 Copenhagen N, Denmark

[7]UNM Comprehensive Cancer Center, 1201 Camino de Salud NE, Albuquerque, NM 87102, USA

[8]Department of Rheumatology and Inflammation Research, Institute of Medicine, Sahlgrenska Academy at University of Gothenburg, Box 480, 40530 Gothenburg, Sweden

*Corresponding authors:

Tudor I. Oprea, MD, PhD: toprea@salud.unm.edu

Gergely Zahoranszky-Kohalmi, PhD: gergely.zahoranszky-kohalmi@nih.gov

Sample Python Code Snippet to Access Neo4COVID19 Database via API

Details on how to install the “py2neo” Python library [1], [2] are provided at https://py2neo.org/v4/.

Sample Python code snippet to connect to the Neo4j database and retrieve the result of the CYPHER query [3].

```
from py2neo import *
graph = Graph(host="neo4covid19.ncats.io", bolt_port=7687, user='', password = '', secure = True)
graph.run("MATCH (t:Target) RETURN t LIMIT 5").data()
```

Furthermore, a script distributed as part of the https://github.com/ncats/neo4covid19 source code repository [4] provides specific examples to query the Neo4COVID19 database. The file is located under `neo4covid19/code/generate_stats.py` where the “`neo4covid19/`” part of the path is the root of the repository.

Pseudo-Code of the Data Integration Workflow

Here we provide the pseudo-code of the data integration workflow conceptualized by *Fig 1*. The name of the variables associated with input data sources is identical to the label of the respective data track.

Algorithm

```
Variable: DataFrame allHostProtein
Variable: DataFrame allPathogenProtein
Variable: DataFrame allDrug
Variable: DataFrame allHPI
Variable: DataFrame allPPI
Variable: DataFrame allDTI


function aggregate_df (DataFrame df, String[] agg_cols, String method) begin

        df = df.aggregate (agg_cols, method)

        return (df)

end function



function aggregate_individual_dataset (DataFrame df, String data_type) begin

        Variable: Dictionary agg_cols{} = {
                "hpi":["pathogen_protein", "host_protein", "interaction", "mechanism"],
                "ppi":["host_protein_a", "host_protein_b", "interaction", "mechanism", "metadata"],
                "dti":["drug_name", "host_protein", "action_type"],
                "host_protein":["host_protein", "activation", "activation_type"],
                "pathogen_protein":["pathogen_protein"],
                "drug":["drug_name"]
        }


        agg_cols = agg_cols[data_type]
```

```
	df = aggregate_df (df, agg_cols, “first”)

	return (df)

end function



function aggregate_datasets (DataFrame df, String data_type) begin

	Variable: agg_cols{} = {
		"hpi":["pathogen_protein", "host_protein"],
		"ppi":["host_protein_a", "host_protein_b"],
		"dti":["drug_name"],
		"host_protein":["host_protein"],
		"pathogen_protein":["pathogen_protein"],
		"drug":["drug_name"]
	}


	agg_cols = agg_cols[data_type]

	df = aggregate_df (df, agg_cols, “concatenate”)

	return (df)

end function



extract_host_proteins (DataFrame df, String data_type) begin

	Variable: DataFrame proteins []
	Variable: String agg_cols[]

	agg_cols = ["host_protein", "data_source"]


	if data_type == "hpi" then
	begin
		proteins = hpi ["host_protein", "data_source", "prioritized_for_pathway_analysis",
"do_ppi_expansion"]
		proteins = aggregate_df (df, agg_cols, “first”)

	else if data_type == "ppi" then
		proteins = ppi ["host_protein_a", "data_source", "prioritized_for_pathway_analysis",
"do_ppi_expansion"]
		proteins = proteins.appendByRow (ppi ["host_protein_b", "data_source"])
```

```
                proteins = aggregate_df (df, agg_cols, “first”)

        else if data_type == "dti" then
                proteins = ppi ["host_protein", "data_source", "prioritized_for_pathway_analysis",
"do_ppi_expansion"]
                proteins = aggregate_df (df, agg_cols, “first”)
        end if

        return (proteins)

end function



extract_pathogen_proteins (DataFrame df) begin

        Variable: DataFrame proteins []
        Variable: String agg_cols[]

        agg_cols = ["host_protein", "data_source"]


        proteins = hpi ["host_protein", "data_source", "prioritized_for_pathway_analysis",
"do_ppi_expansion"]
        proteins = aggregate_df (df, agg_cols, “first”)

        return (proteins)

end function



extract_drugs (DataFrame df) begin

        Variable: DataFrame drugs []
        Variable: String agg_cols[]

        agg_cols = ["host_protein", "data_source"]
        drugs = df["drug_name"]
        proteins = aggregate_df (df, agg_cols, “first”)

        return (drugs)

end function
```

```
function neo4covid19 (DataFrame data_registry)
begin

        Variable: String VIHPs[]
        Variable: String proteins_for_sg[]
        Variable: String proteins_for_string[]


        Variable: DataFrame PPI_SG
        Variable: DataFrame PPI_STRING
        Variable: DataFrame DTI_DC

        for each row in data_registry begin
                df = read(row["filename"])

                // function that performs resource specific data harmonization implemented by the
        investigator
                df = standardize (df, row["harmonization_schema"])

                df = aggregate_individual_dataset (df, row["data_type"])

                if row["data_type"] == "host_protein":
                        allHostProtein = allHostProtein.appendByRow (df)

                else if row["data_type"] == "hpi":
                        allHPI = allHPI.appendByRow (df)
                        allHostProtein = allHostProtein.appendByRow (extract_host_proteins(df,
"hpi"))
                        allPathogenProtein = allPathogenProtein.appendByRow
(extract_pathogen_proteins(df))

                else if row["data_type"] == "dti":
                        allDTI = allDTI.appendByRow (df)
                        allHostProtein = allHostProtein.appendByRow (extract_host_proteins(df, "dti"))
                        allDrug = allDrug.appendByRow (extract_drugs(df))

        end for

        proteins_for_sg = unique(allHostProtein[prioritized_for_ppi_analysis == True]["host_protein"])


        PPI_SG = do_smartgraph_analysis (proteins_for_sg, VIHPs)
                PPI_SG = standardize (PPI_SG, "smart_graph")
        PPI_SG = aggregate_individual_dataset (PPI_SG, "ppi")

        allPPI = PPI_SG


        proteins_for_string = unique(allHostProtein[do_ppi_expansion == True]["host_protein"]
```

```
VIHPs = unique(allHostProtein["host_protein"] - proteins_for_string)

// function to do assemble a PPI subnetwork from STRING induced byhost proteins
PPI_STRING = do_string_expansion (proteins_for_string)

PPI_STRING = standardize (PPI_SG, "string")
PPI_STRING = aggregate_individual_dataset (PPI_STRING, "ppi")

allPPI = allPPI.appendByRow (PPI_STRING)

allHostProtein = allHostProtein.appendByRow (extract_host_proteins(PPI_SG, "ppi"))
allHostProtein = allHostProtein.appendByRow (extract_host_proteins(PPI_STRING, "ppi"))


allHostProtein = aggregate_datasets (allHostProtein, "host_protein")
allPathogenProtein = aggregate_datasets (allPathogenProtein, "pathogen_protein")

allPPI = aggregate_datasets (allPPI, "ppi")
allHPI = aggregate_datasets (allHPI, "hpi")



// function to extract DTIs from DrugCentral based on the implicated host proteins
DTI_DC = getDTIsFromDrugCentral (allHostProtein)

allDTI = allDTI.appendByRow (DTI_DC)

allDTI = aggregate_datasets (allDTI, "dti")


allDrug = extract_drugs (allDTI)


// function that assigns TDL category for host proteins
allHostProtein = annotateTDL (allHostProtein)


// function that cross references PPIs to a reference PPI database, such as Reactome
allPPI = crossReferencePPIs (allPPI)


// function that builds the Neo4j database from the provided arguments
buildNeo4jDatabase(allHostProtein, allPathogenProtein, allDrug, allPPI, allHPI, allDTI)

end function
```

Reproducing the Integration Workflow

A detailed description regarding the compilation of the Neo4COVID19 database is provided at https://github.com/ncats/neo4covid19/blob/master/README.md [1]. Besides the compilation process, the description includes instructions for setting up the necessary environment.

In order to reproduce the workflow, provided the required Python [1] environment has been set up, a local copy of the `neo4covid19` repository needs to be created as follows.

```
git clone https://github.com/ncats/neo4covid19
```

Note, that paths referring to files in this manuscript start with "`neo4covid19`". In this context, `neo4covid19` points to the root directory of the local copy of the cloned repository.

The first stage of the workflow is executed as:

```
python harmonize.py
```

This is followed by assembling the SmartGraph subnetwork. For details, please refer to section "*Assembly of the SmartGraph Subnetwork*".

Once a subnetwork was assembled with the help of SmartGraph, process the results as:

```
python process_sg.py
```

The last stage of the workflow is executed as:

```
python compile.py
```

Assembly of the SmartGraph Subnetwork

In order to reveal potential connection between histone acetyltransferases (HATs) and SARS-CoV-2 virus implicated host proteins (VIHPs), we performed network analysis with the help of the SmartGraph platform [5]. Since a set of VIHPs is compiled in the integration workflow, it was necessary to implement a breakpoint in the workflow. Upon completion of the first part of the workflow, SmartGraph analysis is performed, and the results are subsequently fed to the second stage of the workflow to finish the integration. While this scenario is not ideal, at the time of the workflow creation, the SmartGraph platform did not provide API access.

The gene names of VIHPs and of HATs were mapped to UniProt IDs [6]–[8] to comply with the SmartGraph input requirements.

Below are listed the detailed steps to assemble the SmartGraph subnetwork. Assuming you have created a local copy of the neo4covid19 repository (see above), perform the following steps:

1. Go to SmartGraph (https://smartgraph.ncats.io).
2. Clear the fields "Start Nodes" and "End Nodes" then click on "clear graph".
3. Copy the IDs in column 'uniprot_id' of file `neo4covid19/data/output/sg_proteins_a.tsv` (note that the "`neo4covid19`" points to the root of the `neo4covid19` repository). Insert this set of UniProt IDs as "Start Nodes" in SmartGraph (https://smartgraph.ncats.io).

4. Copy the UniProt IDs from the output of Step 1 located at `neo4covid19/data/output/sg_proteins_b.tsv`. Copy the UniProt IDs and insert them as "End Nodes" in SmartGraph.

5. Set the "Max Distance" parameter to 3.
6. Leave the "PPI Confidence Level" to its default value, i.e. 0.00.
7. Click on "find shortest path".
8. Once the network is assembled in SmartGraph, click on "Download graph", select "Cytoscape JSON", then rename the downloaded file to `SG_HATs_dist_3_conf_0.00.json` and place the file into `neo4covid19/data/input/`.
9. Repeat steps 2-7 but this time use the HATs as “End Nodes” and the UniProt IDs in `neo4covid19/data/output/unique_host_proteins_prestring.txt` as “Start Nodes”.
10. Save the resultant network in "Cytoscape JSON" format and save it as `SG_HATs_reverse_dist_3_conf_0.00.json` and place the file into `neo4covid19/data/input/`.

Expansion of PPIs via StringApp API

Expanding the PPIs present in a preliminary Neo4COVID-19 network was performed in a two-step procedure employing the STRING [9] and stringApp APIs [10].

In the first step, the gene symbols of human proteins in pre-expanded Neo4COVID-19 network were translated into the STRING database identifiers with the STRING API. We utilized the following URL for this API call: https://string-db.org/api/tsv-no-header/get_string_ids . Gene symbols were passed to parameter `identifiers` as a newline "\n" separated string (without quotation marks). Mapping of gene identifiers was forced to a one-to-one mapping by selecting the "best" STRING ID for a given gene symbol by setting `limit` to 1. In addition, we limited the mapping to human genes only by setting `species` to 9606; we included the original IDs in the results by setting `echo_query` to 1; and we provided a string to our liking for `caller_identity`.

Next, with the returned STRING database IDs we made a second API call to URL `https://api.jensenlab.org/network` . The STRING database IDs were passed to the `entities` parameter as a newline "`\n`" separated string. The `additional` parameter was set to 100, which defines the maximal number of proteins the original network can be extended with. Parameter `alpha` was set to its default value of 0.5.

The basis of the expansion is the computation of a connectivity score for proteins not in the query network. The connectivity score is a ratio of the total connectivity score of a given protein to the query proteins versus its total connectivity score to all proteins in STRING database [Ref].

For more details, please refer to the section “Network Expansion” in the study of Doncheva *et al*. [10].

Of note, the following genes present in the pre-extension network were excluded from the STRING extension process as they produced errors when included into the API call: ELOC, EP300, SLC25A5, TUBA1A, STAT1, ELOB, RBX1, CREBBP, SKP1.

Applying Custom Visual Style to the Imported Network in Cytoscape

Instructions below are provided for Cytoscape v3.8.2 [11]. The file containing the custom Cytoscape visual style (style_Neo4COVID19.xml) is distributed as part of the Neo4COVID19 code repository (`neo4covid19/code/style_Neo4COVID19.xml`) [4]. The process of importing and applying the custom style is shown on Fig S2.

Mapping of Viral Gene Names

We have established a mapping between the viral gene names predicted by P-HIPSTer [12], [13] and those reported in the interactome study by [14], [15] The mapping is provided on sheets “ID_Mapping” and “Sheet1_MappedIDs” in the file `data/output/Merged.xlsx` in the `neo4covid19` repository [4].

Reproducing the Use Cases

Instructions below are provided for Cytoscape v3.8.2 [11] with Cytoscape Neo4j Plugin v0.4 [16].

1. Network assembly
   - Establish network connection:

     `Apps > Cypher Queries > Connect to Neo4j Instance`

     Provide `aspire.covid19.ncats.io:7687` as `Hostname`, leave rest of the form empty, then click on `Connect`.
   - Import bipartite HPI network

     `Apps > Cypher Queries > Import Cypher Query`

     Enter this Cypher Query:

     ```
     match (n)-[r:HPI]->(m) return n,r,m
     ```

     Click on `Execute Query`.

2. Apply visual style
   - Please refer to *"Applying Custom Visual Style to the Imported Network in Cytoscape"* section in SI.
3. Topology analysis
   - `Tools -> Analyze Network ...`

     Check in the checkbox next to "Analyze as Directed Graph?",

     click on `OK`
4. Adjust node size as a function of "EdgeCount"
   - Click on `Style` on the left panel and select `Neo4COVID19` in the drop-down box.
   - Click on `Node` on the bottom of the visualization panel.
   - Select `Size`, set `Column` to `EdgeCount`, then set `Mapping` to `Continuous Mapping`.

- Adjust the gradient as shown on the small panel until there is a good separation between low and high degree nodes.

Figures

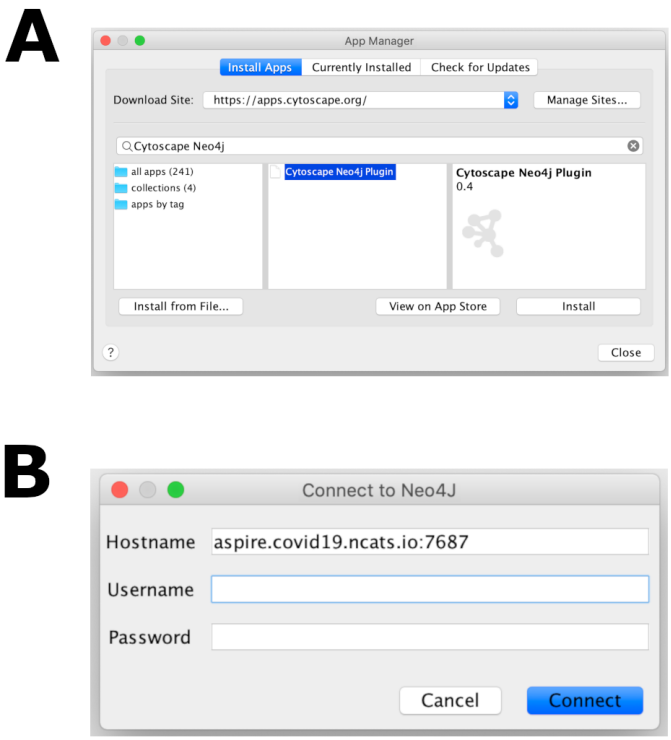

Figure S1. Importing Neo4COVID19 graph database into Cytoscape – part 1. A) Installing the "Cytoscape Neo4j Plugin" [16] by navigating to `"Apps -> App Manager…"`, typing "Cytoscape Neo4j Plugin" in the search bar, selecting the plugin from the results and finally clicking `"Install"`. **B)** Establishing Neo4j Bolt connection ("`Apps > Cypher Queries > Connect to Neo4j Instance`"). Note, that neither username nor password is required. Host: `aspire.covid19.ncats.io:7687` . Screenshots were made from the Cytoscape application.

Figure S2. Importing Neo4COVID19 graph database into Cytoscape – part 2. A) Cypher query to import the entire Neo4COVID19 network into Cytoscape ("`Apps > Cypher Queries > Import Cypher Query`", query: `match (n)-[r]->(m) return n,r,m`) . **B)** Resultant network (after applying the custom visual settings). Nodes representing host and viral proteins, and drugs are denoted by circle, "V", and diamond shaped nodes, respectively. Where applicable, the target development category (TDL) [17], [18] of proteins are color-coded according to legend. Screenshots were made from the Cytoscape application.

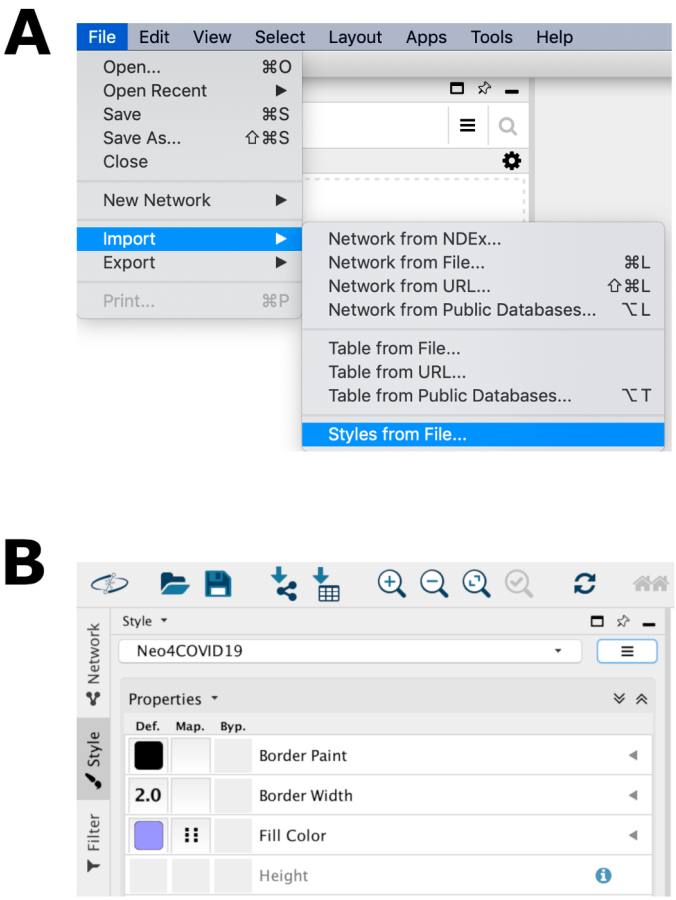

Figure S3. Customizing network visualization. A) Importing the "style_Neo4COVID19.xml" file that contains the custom visual style definition. **B)** Applying the custom visual style "Neo4COVID19". Screenshots taken from Cytoscape 3.8.2.

Tables

Field name	Nodes			Edges		
	Drug	Host Protein	Pathogen Protein	DTI	HPI	PPI
abbreviated_data_source	X	X	X	X	X	X
acquisition_method	X	X	X	X	X	X
activation		X	X			
activation_type		X	X			
CAS_RN	X					
data_source	X	X	X		X	X
directed				X	X	X
drug_name	X					
edge_label				X	X	X
edge_type				X	X	X
gene_symbol		X				
inchi	X					
inchi_key	X					
interaction					X	X
is_activity_known				X		
is_experimental	X	X	X	X	X	X
mechanism					X	X
metadata	X	X	X	X	X	X
name	X	X	X			
node_type	X	X	X			
ns_inchi_key	X					
p_chembl				X		
ref_annotation						X
ref_direction						X
ref_interaction						X
ref_score						X
smiles	X					
source_node				X	X	X
source_node_uuid				X	X	X
source_specific_score		X	X	X	X	X
source_specific_score_type		X	X	X	X	X
target_node				X	X	X
target_node_uuid				X	X	X
tdl		X	X			
uniprot		X				
uuid	X	X	X	X	X	X
is_in_dti_dc	X			X		
is_in_ppi_sg		X				X
is_in_ppi_string		X				X
is_in_dti_jm_cam	X	X		X		
is_in_dti_jm_hcq	X	X		X		
is_in_dti_jm_nhc	X	X		X		
is_in_hostprot_crispr		X				
is_in_hostprot_hats		X				
is_in_hostprot_nat		X				
is_in_hostprot_taiml		X				
is_in_hpi_krogan		X	X		X	
is_in_hpi_phipster		X	X		X	

Table S1. Node and edge attributes of the Neo4COVID19 graph database. Fields highlighted by gray color were derived from the integrated data sources automatically.

Field Code Changed

Integration of Additional Datasets via Data Registry Mechanism

The workflow of this study provides a mechanism that facilitates the extension of the Neo4COVID19 by additional datasets. The mechanism is comprised of maintaining a "registry" file of datasets and an internal standard structure of various type of data, i.e. PPIs, HPIs, DTIs, host and pathogen proteins, and drugs. Of note, addition of PPIs (from STRING [9]), and DTIs and drug information (DrugCentral [19]) is already integrated into the workflow. Furthermore, data sources focused on pathogen proteins can be added as HPI, where the host protein might be defined as an unknown host protein.

The mechanism currently supports the semi-automated extension of the Neo4COVID19 database by three types of data Namely, HPIs, host protein targets and drug-target interactions which datatypes, be it experimental or predicted, are expected to emerge rapidly in the case of a pandemic. Accordingly, additional datasets need to be included into the file that is distributed with the repository (please refer to section "Reproducing the Integration Workflow" in order to interpret this file location correctly):

```
neo4covid19/input/data_source_registry.txt .
```

The file contains several fields that need to be specified by the investigator in connection with each dataset to be integrated, see *Table S2* for details. Once a new dataset was registered in the file, a new if statement needs to be added in the `standardize_data()` function of the `neo4covid19/code/harmonize.py` source code file. Subsequently, a new function needs to be created

in `neo4covid19/code/standardize.py` that converts the respective input to the standard internal format of the corresponding datatype. Note, that the "`abbreviated_data_source`" information will be automatically converted to Boolean database fields in Neo4j. In this process, each unique value of this field in the context of a specific data type will become a Boolean field. These fields allow for the facile filtering of data sources based on data provenance.

Additional data sources beyond the currently supported host protein, HPI, and DTI data types might be integrated into the workflow via bypassing the data registry mechanism. However, this will require substantial modification of the source the discussion of which is outside of the scope of this study. Nonetheless, in *Table S3* we provide the specifications for the standardized internal data formats of all data basic types used to build the Neo4COVID19 database, namely HPI, PPI, DTI, host protein, pathogen protein and drug types. This information can provide guidance for the implementation of new code logic that integrates potentially any arbitrary data sets as long as they can be converted to the specified format [20].

Field Name	Field Value Syntax
input	filename if input is TAB-separated file, or filename;sheet_name if input is an Excel (.xlsx) file
data_type	hpi / host_protein / dti
is_experimental	TRUE / FALSE
data_source	free text: short descritption of datasource
abbreviated_data_source	free text: abbreviation of datasource, suggestion: indicate dataype in abrevviation, e.g. hpi_mydata, hostprot_mydata2, dti_mylab
original_score	na/column_name (column_name: name of column in input file containing the score/confidence value associated with the data type)
original_score_type	na/score_type (score_type, free text, indicating the type of original_score, e.g. P-value, confidence, p_chembl, etc.)
prioritized_for_pathway_analysis	TRUE/FALSE (host proteins flagged as TRUE in this field will be treated as starting nodes in SmartGraph pathway analysis, and all other host proteins upstream of the STRING expansion in the workflow will be treated as destination targets in SmartGraph, and *vice versa*)
do_ppi_expansion	TRUE / FALSE
harmonization_schema	string: you will need to add this harmonization scheme into the harmonize.py and implement the logic accordingly in standardize.py, example values: dh_dti_mylab, dh_targets, etc.
acquisition_method	free text: describing how the data was acquired
metadata_columns	semicolon separated list of column names in input file which should be converted into metadata
is_directed	TRUE / FALSE / na (use “na” for host protein target data type)

Table S2. Structure of data registry file. Field names, data types and indicated TRUE/FALSE and “na” values are case-sensitive.

	Host Protein	HPI	DTI	PPI	Pathogen Protein	Drug
host_protein (string, gene symbol)	m	m	m			
pathogen_protein (string, name)		m			m	
host_protein_a (string, gene_symbol)				m		
host_protein_b (string, gene_symbol)				m		
drug_name (string)			m			m
interaction (string)		m		m		
mechanism (string)		m		m		
action_type (string)			m			
p_chembl (string)			m			
directed (bool)		m	m	m		
metadata (string, format: key1:value1;key2:value2)	o / na	o / na	o / na	o / na	o / na	o / na
abbreviated_data_source (string)	m	m	m	m	m	m
is_experimental (bool)	m	m	m	m	m	m
data_source (string)	m	m	m	m	m	m
acquisition_method (string)	m	m	m	m	m	m
prioritized_for_pathway_analysis (bool)	m	m	m	m	m	
do_ppi_expansion (bool)	m	m	m	m	m	
source_specific_score (string)	o / na	o / na	o / na	o / na	o / na	
source_specific_score_type (string)	o / na	o / na	o / na	o / na	o / na	
activation (string)	o / na				o / na	
activation_type (string)	o / na				o / na	
smiles (string)						o / na
inchi (string)						o / na
inchi_key (string)						o / na
ns_inchi_key (string)						o / na
CAS_RN (string)						o / na

Table S3. Definition of mandatory and optional fields of internal data structure.

Abbreviations: “m”: mandatory, “o / na”: optional, or fill with literal string “na” if not applicable/available. Empty fields should not be included in a given type of data structure. Data types shaded by gray are not yet supported via the data registry mechanism, but such types of data can be added to the workflow via the source code, provided that the indicated data structure is observed. However, PPI and drug information are automatically pulled in from pre-defined sources in the workflow, and pathogen proteins can be added as HPIs via the data registry mechanism. Field names are case-sensitive.

Field Code Changed

References

[1] "Python Core Team. Python: A Dynamic, Open Source Programming Language. Python Software Foundation." https://www.python.org/.

[2] "Python Library 'py2neo' v4." .

[3] "Neo4j Graph Database." https://neo4j.com/.

[4] "Code Repository 'neo4covid19.'" https://github.com/ncats/neo4covid19.git.

[5] G. Zahoránszky-Kőhalmi, T. Sheils, and T. I. Oprea, "SmartGraph: A Network Pharmacology Investigation Platform," *J. Cheminform.*, vol. 12, no. 1, p. 5, Dec. 2020, doi: 10.1186/s13321-020-0409-9.

[6] "UniProt: A Worldwide Hub of Protein Knowledge," *Nucleic Acids Res.*, vol. 47, no. D1, pp. D506–D515, Jan. 2019, doi: 10.1093/nar/gky1049.

[7] T. U. Consortium, "UniProt: The Universal Protein Knowledgebase," *Nucleic Acids Res.*, vol. 45, no. D1, pp. D158–D169, 2016, doi: 10.1093/nar/gkw1099.

[8] "UniProt Proggrammatic Service for ID Mapping." https://ftp.uniprot.org/pub/databases/uniprot/current_release/knowledgebase/idmapping/by_organism/HUMAN_9606_idmapping.dat.gz.

[9] D. Szklarczyk *et al.*, "STRING v11: Protein–Protein Association Networks With Increased Coverage, Supporting Functional Discovery in Genome-Wide Experimental Datasets," *Nucleic Acids Res.*, vol. 47, no. D1, pp. D607–D613, Jan. 2019, doi: 10.1093/nar/gky1131.

[10] N. T. Doncheva, J. H. Morris, J. Gorodkin, and L. J. Jensen, "Cytoscape StringApp:

Network Analysis and Visualization of Proteomics Data," *J. Proteome Res.*, vol. 18, no. 2, pp. 623–632, Feb. 2019, doi: 10.1021/acs.jproteome.8b00702.

[11] P. Shannon, "Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks," *Genome Res.*, vol. 13, no. 11, pp. 2498–2504, Nov. 2003, doi: 10.1101/gr.1239303.

[12] G. Lasso *et al.*, "A Structure-Informed Atlas of Human-Virus Interactions," *Cell*, vol. 178, no. 6, pp. 1526-1541.e16, Sep. 2019, doi: 10.1016/j.cell.2019.08.005.

[13] "P-HIPSTer." http://phipster.org/.

[14] D. E. Gordon *et al.*, "A SARS-CoV-2 Protein Interaction Map Reveals Targets for Drug Repurposing," *Nature*, vol. 583, no. 7816, pp. 459–468, Jul. 2020, doi: 10.1038/s41586-020-2286-9.

[15] Krogan, "A SARS-CoV-2 Protein Interaction Map Reveals Targets for Drug Repurposing," [Online]. Available: https://www.biorxiv.org/content/10.1101/2020.03.22.002386v3.

[16] S. Warris, S. Dijkxhoorn, T. van Sloten, and B. van de Vossenberg, "Mining Functional Annotations Across Species," *bioRxiv*, 2018, doi: 10.1101/369785.

[17] D.-T. Nguyen *et al.*, "Pharos: Collating Protein Information to Shed Light on the Druggable Genome," *Nucleic Acids Res.*, vol. 45, no. D1, pp. D995–D1002, Nov. 2016, doi: 10.1093/nar/gkw1072.

[18] T. I. Oprea *et al.*, "Unexplored Therapeutic Opportunities in the Human Genome," *Nat. Rev. Drug Discov.*, vol. 17, p. 317, Mar. 2018, [Online]. Available: https://doi.org/10.1038/nrd.2018.14.

[19] O. Ursu *et al.*, "DrugCentral 2018: An Update," *Nucleic Acids Res.*, vol. 47, no. D1, pp.

D963–D970, Jan. 2019, doi: 10.1093/nar/gky963.

[20] The Authors are thankful for the suggestions of Reviewer 2 regarding improving the workflow by the addition of data source “plug-in” mechanism and more robust data standardization.