nature portfolio

https://doi.org/10.1038/s41596-022-00797-1

Supplementary information

Mapping the common gene networks that underlie related diseases

In the format provided by the authors and unedited

Supplementary Procedure

The procedure included here matches the main text, but here we include the code blocks. Steps 1-24 are demonstrated in an example notebook: https://github.com/ucsd-ccbb/NetColoc/blob/main/example_notebooks/ASD_CHD_NetColoc_analysis.ipynb

The remaining steps 25-30 should be carried out in NDEx/Cytoscape.

Obtain input gene sets and gene network

1. Load required packages.
   ?Troubleshooting

```
# load required packages
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import networkx as nx
import pandas as pd
import re
import random

from IPython.display import display

import getpass
import ndex2

import json
import cdapsutil

from gprofiler import GProfiler
gp = GProfiler("MyToolName/0.1")

from scipy.stats import hypergeom
from scipy.stats import norm

# latex rendering of text in graphs
import matplotlib as mpl
mpl.rc('text', usetex = False)
mpl.rc('font', family = 'serif')

from matplotlib import rcParams

# set plotting properties
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Arial']
sns.set(font_scale=1.4)
sns.set_style('white')
sns.set_style("ticks", {"xtick.major.size": 15, "ytick.major.size": 15})
plt.rcParams['svg.fonttype'] = 'none'
```

```
from datetime import datetime
import sys
%matplotlib inline
# verify DDOT was installed
import ddot
from netcoloc import netprop_zscore, netprop, network_colocalization, validation
```

2. Select two gene sets of interest. Load gene sets from text files into Python. These gene sets should contain between 5 and 500 genes and come from experimental data, rather than manual curation, to avoid bias.
   OPTIONAL: In some use cases, the gene sets of interest may accompany a score (such as p-value, or log fold change, in an RNA-Seq differential expression experiment). For these use cases, we provide an optional step to aid the researcher in finding an optimal choice of threshold by sweeping over a range of filtering criteria to maximize the observed divided by expected network intersection size. The genes which meet these criteria should be retained for use in the following steps. This process is illustrated in an example notebook https://github.com/ucsd-ccbb/NetColoc/blob/main/example_notebooks/Evalute_scored_input_gene_lists.ipynb.

```
# set names of geneset 1 and geneset 2
# ------ customize this section based on your gene sets and how they should be
# labeled -------
d1_name='ASD'
d2_name='CHD'
D1_df = pd.read_csv('data/Satterstrom--Top-102-ASD-genes--May2019.csv')
D1_df.index = D1_df['gene']
print('Number of '+d1_name+' genes:', len(D1_df))
D1_genes = D1_df.index.tolist() # define rare variant genes to seed network
propagation
print('First 5 genes:',  D1_genes[0:5])

D2_df = pd.read_csv('data/CHD_HC.tsv',sep='\t')

D2_genes = D2_df['0'].tolist()
print('Number of '+d2_name+' genes:', len(D2_genes))
print('First 5 genes:', D2_genes[0:5])

# Output the overlap between the two gene sets.
print('Number of '+d1_name+' and '+d2_name+' genes overlapping:',
len(np.intersect1d(D1_genes,D2_genes)))
```

3. Select a gene interaction network to use for the analysis. Identify the network UUID on NDEx[27] and use this to import to a Jupyter notebook. We recommend using PCNet as a starting point, but a user may want to switch to “STRING high confidence” if using a machine with low memory (< 8GB RAM).
   ?Troubleshooting

```
interactome_uuid='4de852d9-9908-11e9-bcaf-0ac135e8bacf' # for PCNet
# interactome_uuid='275bd84e-3d18-11e8-a935-0ac135e8bacf' # for STRING high
```

```
confidence
ndex_server='public.ndexbio.org'
ndex_user=None
ndex_password=None
G_int = ndex2.create_nice_cx_from_server(
            ndex_server,             username=ndex_user,
            password=ndex_password,
            uuid=interactome_uuid
        ).to_networkx()
nodes = list(G_int.nodes)

# remove self edges from network
G_int.remove_edges_from(nx.selfloop_edges(G_int))

# print out the numbers of nodes and edges in the interactome for diagnostic
purposes:
print('Number of nodes:', len(G_int.nodes))
print('\nNumber of edges:', len(G_int.edges))
int_nodes = list(G_int.nodes)
```

Identify subnetworks of colocalized genes

4. Precalculate matrices needed for network propagation, using the functions `netprop.get_normalized_adjacency_matrix` and `netprop.get_individual_heats_matrix`, referred to as w' and w'' in the following. This step will take a few minutes (more for denser networks). A benchmarking analysis demonstrates that the runtime required scales with the number of edges (w') and the number of nodes (w'') (**Supplementary Figure 2a,b).** If the researcher plans on running multiple analyses they may find it useful to save these matrices. We include instructions for saving and reloading. We caution that, because these matrices are not sparse, saving and reloading can take a few minutes, and the saved file can be a few GB, so for many networks it may be faster to recompute the matrices each time. The diffusion parameter, which controls the rate of propagation through the network, may be set in this step. In practice, we have found that results are not dependent on the choice of this parameter (**Supplementary Figure 3**), and recommend using the default value of 0.5. ?Troubleshooting

```
# pre-calculate matrices used for network propagation. this step takes a few
minutes, more for denser interactomes
print('\ncalculating w_prime')
w_prime = netprop.get_normalized_adjacency_matrix(G_int, conserve_heat=True)
print('\ncalculating w_double_prime')
w_double_prime = netprop.get_individual_heats_matrix(w_prime, .5)

# w_double_prime can be saved to the file netcoloc_w_double_prime.npy
# in current working directory with the following call:
#
# np.save('netcoloc_w_double_prime.npy', w_double_prime)

# and reloaded later with:
```

```
# w_double_prime = np.load('netcoloc_w_double_prime.npy')
#
# NOTE: Saving w_double_prime results in a several gigabyte file and
#       takes a minute or more to save and load
```

5. Subset input genes sets to genes found in the selected network. Only genes contained in the interaction network will be retained as “seed” genes for downstream analysis. ?Troubleshooting

```
# subset seed genes to those found in interactome
print("Number of D1 genes:", len(D1_genes))
D1_genes = list(np.intersect1d(D1_genes,int_nodes))
print("Number of D1 genes in interactome:", len(D1_genes))

print("Number of D2 genes:", len(D2_genes))
D2_genes = list(np.intersect1d(D2_genes,int_nodes))
print("Number of D2 genes in interactome:", len(D2_genes))
```

6. Compute network proximity scores from both seed gene sets,z_1 and z_2, independently, using the function `netprop_zscore.calculate_heat_zscores`. The network proximity scores include a correction for the degree distribution of the input gene sets (**Supplementary Figure 4**). The runtime required for computing the network proximity scores increases linearly with the number of nodes in the underlying interaction network and with the size of the input gene list (**Supplementary Figure 2c)**.

```
# D1 network propagation
print('\nCalculating D1 z-scores: ')
z_D1, Fnew_D1, Fnew_rand_D1 = netprop_zscore.calculate_heat_zscores(w_double_prime,
int_nodes,dict(G_int.degree),D1_genes, num_reps=1000,minimum_bin_size=100)

z_D1 = pd.DataFrame({'z':z_D1})
z_D1.sort_values('z',ascending=False).head()
```

```
# D2 network propagation
print('\nCalculating D2 z-scores: ')
z_D2, Fnew_D2, Fnew_rand_D2 = netprop_zscore.calculate_heat_zscores(w_double_prime,
int_nodes, dict(G_int.degree),D2_genes, num_reps=1000, minimum_bin_size=100)

z_D2 = pd.DataFrame({'z':z_D2})
z_D2.sort_values('z',ascending=False).head()
```

7. Build the NetColoc subnetwork and evaluate it for significant network colocalization. To build the NetColoc subnetwork, we take the product of the two proximity vectors as follows:

$$z_{coloc} = z_1 * z_2$$

We then select genes with z_{coloc} greater than a threshold ($z_{coloc} > 3$ default, but can be set by the user), and network proximity scores individually larger than a nominal threshold ($z_1 > 1.5$, and $z_2 > 1.5$ default, but can be set by the user). The genes meeting these criteria and associated interactions make up the network colocalization subnetwork. The authors have found that the default threshold values work well in practice to find the set of genes that is proximal to both seed gene sets. Tuning them higher will lead to fewer false positives but more false negatives. Similarly, tuning them lower will lead to more false positives but fewer false negatives. Either may be warranted given the specifics of an experiment. The researcher may conduct a sensitivity analysis of these thresholds to find a balance between a higher NetColoc enrichment score, but smaller network, and a lower NetColoc enrichment score, but larger network (**Supplementary Figure 5)**. The function `network_colocalization.calculate_network_enrichment` is provided to enable such a sensitivity analysis. In this function, the network colocalization score is computed for the gene set pair, based on the observed network overlap and expected network overlap from a null distribution, over a range of z-score thresholds. We recommend using the default thresholds unless the use case calls for higher or lower stringency. Choosing the thresholds which optimize the network colocalization score risks leaving out important phenotype-related genes. If gene sets are significantly colocalized, proceed with the analysis. Gene sets that are not significantly colocalized in the network have no evidence for shared underlying pathways, and thus proceeding with an analysis of the network intersection in this case is not likely to return meaningful results.

```
zlist = [2,3,4,5,9,12]
z12list = [1,1.5,2]
netcoloc_enrichment_df =
network_colocalization.calculate_network_enrichment(z_D1,z_D2,zthresh_list =
zlist,z12thresh_list=z12list,verbose=True)

netcoloc_enrichment_df.head()
```

```
# Plot the sizes of the observed and expected NetColoc subnetworks.

plt.figure(figsize=(5,5))
sns.lineplot(x='z_comb',y='obs_exp',data=netcoloc_enrichment_df,hue='z_12',style='z_1
2',markers=['o','o','o'],dashes=False,palette='viridis')

ylim_max=plt.gca().get_ylim()[1]
plt.ylim([1,ylim_max])
plt.plot([3,3],[1,ylim_max],'k--',alpha=.5)

plt.xlabel('z threshold')
plt.ylabel('observed/expected \nnetwork intersection size')
plt.legend(title="$z_{1,2}$")

plt.figure(figsize=(5,5))
sns.lineplot(x='z_comb',y='observed_overlap',data=netcoloc_enrichment_df,hue='z_12',s
```

```
tyle='z_12',markers=['o','o','o'],dashes=False,palette='viridis')

ylim_max=plt.gca().get_ylim()[1]
plt.ylim([0,ylim_max])
plt.plot([3,3],[0,ylim_max],'k--',alpha=.5)

plt.xlabel('z threshold')
plt.ylabel('number of genes in \nnetwork intersection')
plt.legend(title="$z_{1,2}$")
```

```
# ----------- select thresholds for NetColoc subgraph -----------------
zthresh=3 # default = 3
z12_thresh=1.5 # default = 1.5
# -------------------------------------------------------------
focal_netcoloc_enrichment =
netcoloc_enrichment_df[(netcoloc_enrichment_df['z_comb']==zthresh) & \

(netcoloc_enrichment_df['z_12']==z12_thresh)].iloc[0]

# plot the observed and expected overlaps
plt.figure(figsize=(2,5))
plt.bar([0,1],[focal_netcoloc_enrichment['expected_overlap_mean'],

focal_netcoloc_enrichment['observed_overlap']],color=['gray','black'],width=1)
plt.errorbar([0],[focal_netcoloc_enrichment['expected_overlap_mean']],

[2*focal_netcoloc_enrichment['expected_overlap_std']],color='k',capsize=4)
plt.ylabel('# NetColoc genes')
plt.xticks([0,1],['expected','observed'],rotation='vertical')
plt.xlim([-1,2])
```

```
# select the genes in the network intersection, make a subgraph

G_overlap =
network_colocalization.calculate_network_overlap_subgraph(G_int,z_D1['z'],z_D2['z'],z
_score_threshold=zthresh,z1_threshold=z12_thresh,z2_threshold=z12_thresh)

print("Nodes in overlap subgraph:", len(G_overlap.nodes()))
print("Edges in overlap subgraph:", len(G_overlap.edges()))
```

8. OPTIONAL: Transform NetColoc subnetwork edges to cosine similarities with the function `network_colocalization.transform_edges`. The cosine similarity score between two genes represents the extent to which those genes have similar interactors. In practice, the cosine similarity transformed score helps to visually reveal the underlying clustering structure present in a network.

```
G_cosSim=network_colocalization.transform_edges(G_overlap,method='cosine_sim',edge_we
ight_threshold=0.95)
```

Compute network colocalized systems map

9. Convert network colocalization subnetwork to form used in community detection module.

```
# compile dataframe of metadata for overlapping nodes
node_df = pd.DataFrame(index=list(G_overlap.nodes))
node_df = node_df.assign(d1_seeds=0, d2_seeds=0, d1_name=d1_name, d2_name=d2_name)
node_df.loc[list(np.intersect1d(seed_dict[d1_name],node_df.index.tolist())),
'd1_seeds']=1
node_df.loc[list(np.intersect1d(seed_dict[d2_name],node_df.index.tolist())),
'd2_seeds']=1
node_df['z_d1']=z_D1.loc[list(G_overlap.nodes)]['z']
node_df['z_d2']=z_D2.loc[list(G_overlap.nodes)]['z']
node_df['z_both']=node_df['z_d1']*node_df['z_d2']
node_df['sum_seeds']=node_df['d1_seeds']+node_df['d2_seeds']

node_df = node_df.sort_values('z_both',ascending=False)
node_df.head(15)
```

10. Run community detection on the NetColoc subnetwork to identify highly interacting subsystems. We recommend using the HiDef clustering algorithm[24] which is included in the community detection package, along with other commonly used clustering algorithms.

```
print("Nodes in overlap subgraph:", len(G_overlap.nodes()))
print("Edges in overlap subgraph:", len(G_overlap.edges()))
# Create cx format of overlap subgraph
G_overlap_cx = ndex2.create_nice_cx_from_networkx(G_overlap)
G_overlap_cx.set_name(d1_name+'_'+d2_name+'_NetColoc_subgraph')
for node_id, node in G_overlap_cx.get_nodes():
    data = node_df.loc[node['n']]
    for row, value in data.items():
        if row == 'd1_seeds' or row == 'd2_seeds' or row=='sum_seeds':
            data_type = 'double'
        elif row=='d1_name' or row=='d2_name':
            data_type='string'
        else:
            data_type = 'double'
        G_overlap_cx.set_node_attribute(node_id, row, value, type=data_type)

cd = cdapsutil.CommunityDetection()

# Run HiDeF on CDAPS REST service
G_hier = cd.run_community_detection(G_overlap_cx,
algorithm='hidefv1.1beta',arguments={'--maxres':'20'})
```

```
# Print information about hierarchy
print('Hierarchy name: ' + str(G_hier.get_name()))
print('# nodes: ' + str(len(G_hier.get_nodes())))
```

```
print('# edges: ' + str(len(G_hier.get_edges())))
```

11. Convert the NetColoc hierarchy to networkx format, and write out features of the hierarchy to a pandas dataframe, for easier manipulation in Python.

```
G_hier = G_hier.to_networkx(mode='default')
G_hier

nodes = G_hier.nodes()

# print the number of nodes and edges in the hierarchy for diagnostic purposes
print('Number of nodes:', len(G_hier.nodes()))
print('\nNumber of edges:', len(G_hier.edges()))
# add node attributes to the dataframe for easier access
hier_df = pd.DataFrame.from_dict(dict(G_hier.nodes(data=True)), orient='index')
hier_df['system_ID']=hier_df.index.tolist()
# some columns are not the right type
hier_df['CD_MemberList_Size']=[int(x) for x in
hier_df['CD_MemberList_Size'].tolist()]
hier_df['HiDeF_persistence']=[int(x) for x in hier_df['HiDeF_persistence'].tolist()]
hier_df.head()
```

12. OPTIONAL: Systems that do not contain any seed genes may be removed in order to focus on systems in which perturbations are known to have an effect.
?Troubleshooting

```
hier_df.index=hier_df['name']
hier_df.head()

num_d1_seeds, num_d2_seeds, num_both_seeds = [],[],[]
frac_d1_seeds, frac_d2_seeds, frac_both_seeds=[],[],[]

systems_keep = []
for c in hier_df.index.tolist():
    system_genes = hier_df['CD_MemberList'].loc[c].split(' ')
    d1_temp = list(np.intersect1d(system_genes,seed_dict[d1_name]))
    d2_temp = list(np.intersect1d(system_genes,seed_dict[d2_name]))
    num_d1_temp = len(d1_temp)
    num_d2_temp = len(d2_temp)
    if (num_d1_temp+num_d2_temp)>0: # keep the system if it has at least 1 seed gene
        systems_keep.append(c)
        num_both_temp = len(np.intersect1d(d1_temp,d2_temp))
        num_both_seeds.append(num_both_temp)
        num_d1_seeds.append(num_d1_temp-num_both_temp)
        num_d2_seeds.append(num_d2_temp-num_both_temp)

        frac_both_seeds.append(num_both_temp/float(len(system_genes)))
        frac_d1_seeds.append((num_d1_temp-num_both_temp)/float(len(system_genes)))
        frac_d2_seeds.append((num_d2_temp-num_both_temp)/float(len(system_genes)))
```

```
frac_no_seeds =
np.subtract(1.0,np.array([frac_d1_seeds,frac_d2_seeds,frac_both_seeds]).sum(axis=0))

hier_df = hier_df.loc[systems_keep]
hier_df['num_d1_seeds']=num_d1_seeds
hier_df['num_d2_seeds']=num_d2_seeds
hier_df['num_both_seeds']=num_d2_seeds
hier_df['frac_d1_seeds']=frac_d1_seeds
hier_df['frac_d2_seeds']=frac_d2_seeds
hier_df['frac_both_seeds']=frac_both_seeds
hier_df['frac_no_seeds']=frac_no_seeds
print('Number of nodes with seed genes:', len(hier_df))

hier_df.head()

# prune G_hier--> only keep systems with at least one seed gene

nkeep=[]
for n in list(G_hier.nodes()):
    if G_hier.nodes(data=True)[n]['name'] in systems_keep:
        nkeep.append(n)


G_hier = nx.subgraph(G_hier, nkeep)
print('Number of nodes with seed genes:', len(G_hier.nodes()))
print('Number of edges remaining:', len(G_hier.edges()))
```

13. OPTIONAL: Sneak peek of the NetColoc hierarchy. Full annotation and visualization are conducted later in the analysis pipeline, but the researcher may find it helpful to get a sense of the size and structure of the NetColoc hierarchy.
?Troubleshooting

```
network_colocalization.view_G_hier(G_hier)
```

14. Annotate systems with gprofiler[10], a functional enrichment tool. Annotate moderately sized systems (between 50 to 1000 genes per system) if the systems are significantly enriched for a Gene Ontology (GO) biological process. To increase the stringency of the annotation, require that the GO term is enriched with $p<1\times10^{-5}$ and shares at least 3 genes with the system. Label the system using the GO term that meets these criteria and has the highest sum of precision and recall. Systems without a GO term meeting these criteria are labeled with their unique system ID.

```
system_name_list = []
for p in hier_df.index.tolist():
    focal_genes=hier_df['CD_MemberList'].loc[p].split(' ')
    print(p)
```

```
    print(len(focal_genes))
    if len(focal_genes)>2:
        gp_temp =
pd.DataFrame(gp.profile(focal_genes,significance_threshold_method='fdr',
                                                sources=['REAC']))
        if len(gp_temp)>0: # make sure data is not empty

            # make sure terms are specific, and overlap with at least 3 genes
            gp_temp = gp_temp[(gp_temp['term_size']<1000)&(gp_temp['term_size']>50)]
            gp_temp = gp_temp[gp_temp['intersection_size']>=3]

            gp_temp = gp_temp[gp_temp['p_value']<1E-5] # set a stringent pvalue
threshold

            gp_temp = gp_temp.sort_values('recall',ascending=False)

            if len(gp_temp)>1:
                system_name_list.append(gp_temp.head(1)['name'].tolist()[0])
            else:
                system_name_list.append(p)
        else:
            system_name_list.append(p)


        display(gp_temp.head())

    else:
        system_name_list.append(p)
```

Validate identified genes and systems

15. Load and parse mouse variant database.

```
mgi_df = validation.load_MGI_mouseKO_data()
mgi_df.head()
```

```
MPO = validation.load_MPO()
```

16. Identify phenotype(s) of interest. We recommend including a negative control, a phenotype that is not expected to overlap with the two phenotypes of interest.

```
def get_MP_description(term_id, ontology=MPO):
    return ontology.node_attr.loc[term_id, "description"]

# find terms related to brain
# ---- modify this part as needed for your project -----
MP_focal_brain_list = []
for t in MPO.node_attr.index.tolist():
    descr_temp = MPO.node_attr.loc[t]['description']
    if descr_temp.find('nervous')>-1:
```

```
        MP_focal_brain_list.append(t)
    elif descr_temp.find('neuron')>-1:
        MP_focal_brain_list.append(t)
    elif descr_temp.find('synapt')>-1:
        MP_focal_brain_list.append(t)

print("Number of brain phenotypes", len(MP_focal_brain_list))
print("Example brain phenotypes:")
print("\n".join([mp+" - "+get_MP_description(mp) for mp in
MP_focal_brain_list[0:10]]))


# find terms related to heart
MP_focal_heart_list = []
for t in MPO.node_attr.index.tolist():
    descr_temp = MPO.node_attr.loc[t]['description']
    if descr_temp.find('cardi')>-1:
        MP_focal_heart_list.append(t)
    elif descr_temp.find('heart')>-1:
        MP_focal_heart_list.append(t)

print("Number of heart phenotypes:", len(MP_focal_heart_list))
print("Example heart phenotypes:")
print("\n".join([mp+" - "+get_MP_description(mp) for mp in
MP_focal_heart_list[0:10]]))
```

17. Compute the enrichment of selected phenotype(s) in the NetColoc subnetwork as a whole to identify the phenotypes with the strongest association with the full NetColoc subnetwork. By computing the enrichment in the entire NetColoc subnetwork, we identify the phenotypes with the strongest association with the entire set of genes identified to be related to both input sets.

```
MP_focal_list = ['MP:0002419']+MP_focal_brain_list
root_KO_brain_df=validation.MPO_enrichment_root(hier_df,MPO,mgi_df,MP_focal_list,G_in
t,verbose=True)
root_KO_brain_df.head()

MP_focal_list = MP_focal_heart_list
root_KO_heart_df=validation.MPO_enrichment_root(hier_df,MPO,mgi_df,MP_focal_list,G_in
t,verbose=True)
root_KO_heart_df.head()

# join brain and heart results together
root_KO_brain_df['MPO_term_type']='brain'
root_KO_heart_df['MPO_term_type']='heart'
root_KO_df = pd.concat([root_KO_heart_df, root_KO_brain_df], axis=0, sort=False)
root_KO_df['MPO_term_type'].loc['MP:0002419']='neg_ctrl'
root_KO_df = root_KO_df.sort_values('OR_p')
root_KO_df.head()

# check the number of terms associated with each type of phenotype
```

```
root_KO_df['MPO_term_type'].value_counts()
```

```
# plot top performing brain and heart terms + negative control term (MP:0002419) (for
terms which have at least 150 genes)

heart_terms_plot = root_KO_heart_df[root_KO_heart_df['num_genes_in_term']>150]
heart_terms_plot =
heart_terms_plot.sort_values('OR_p',ascending=True).head(5).index.tolist()

brain_terms_plot = root_KO_brain_df[root_KO_brain_df['num_genes_in_term']>150]
brain_terms_plot =
brain_terms_plot.sort_values('OR_p',ascending=True).head(5).index.tolist()

neg_ctrl_terms_plot=['MP:0002419']

terms_plot = brain_terms_plot+heart_terms_plot +neg_ctrl_terms_plot
plt.figure(figsize=(3,6))

plt.errorbar(root_KO_df.loc[terms_plot]['log_OR'],np.arange(len(terms_plot)),

xerr=[np.subtract(root_KO_df.loc[terms_plot]['log_OR'],root_KO_df.loc[terms_plot]['lo
g_OR_CI_lower']),

np.subtract(root_KO_df.loc[terms_plot]['log_OR_CI_upper'],root_KO_df.loc[terms_plot][
'log_OR'])],color='k',fmt='o')

color_temp =
root_KO_df.loc[terms_plot]['MPO_term_type'].map({'brain':'blue','heart':'red','neg_ct
rl':'black'})

sns.scatterplot(x=root_KO_df.loc[terms_plot]['log_OR'],

y=np.arange(len(terms_plot)),size=root_KO_df.loc[terms_plot]['num_genes_in_term'],
sizes=(200, 2000), alpha=.5, hue=color_temp.tolist(),
palette={'blue':'blue','red':'red','black':'black'},legend=False)

plt.yticks(np.arange(len(terms_plot)),root_KO_df.loc[terms_plot]['MP_description'])
plt.xticks([0,1,2])
plt.xlabel('log(OR) +- 95% CI')

plt.plot([0,0],[-.8,len(terms_plot)-.5],'--',color='gray')
plt.ylim([-0.8,len(terms_plot)-.5])

plt.gca().invert_yaxis()
```

18. Compute the enrichment of phenotype(s) in NetColoc subsystems. Some phenotypes may have stronger associations with NetColoc subsystems than with the full subnetwork. In this step, we calculate the enrichment of selected phenotypes in each NetColoc subsystem.

```
MP_focal_top = root_KO_df.head(10).index.tolist() # record the top 10 overall
MP_full_results_df =
validation.MPO_enrichment_full(hier_df,MPO,mgi_df,MP_focal_top,G_int)
MP_full_results_df.head()

print("Top ten terms:")
print("\n".join([mp+" - "+get_MP_description(mp) for mp in MP_focal_top]))
```

19. Annotate the NetColoc systems map with mouse variant data, input genes and enriched GO terms.

```
# add the best gprofiler annotation
MP_full_results_df['gprofiler_name']=pd.Series(system_name_list,
index=hier_df.index.tolist())
# don't annotate the root node
root_node =
hier_df['CD_MemberList_Size'].sort_values(ascending=False).head(1).index.tolist()[0]
MP_full_results_df.loc[root_node, 'gprofiler_name']=d1_name+'-'+d2_name+' systems
map'

# also add the frac_seeds/num_seeds data here
MP_full_results_df=MP_full_results_df.join(hier_df[['num_d1_seeds','num_d2_seeds','fr
ac_d1_seeds','frac_d2_seeds','frac_both_seeds','frac_no_seeds']], how='left')

MP_full_results_df.head()
```

20. Export the NetColoc systems map to NDEx with the default style. Default style maps the fraction of seed genes from input set 1 (red) and input set 2 (blue) to node pie charts. The remaining white fraction indicates the fraction of genes in each system that are not in either input set but that are implicated by network propagation.

```
# Convert G_hier to nice cx network
node_id_to_node_name = nx.get_node_attributes(G_hier, 'name')
for node_id in list(G_hier.nodes):
    del G_hier.nodes[node_id]['name']

G_hier_cx = ndex2.create_nice_cx_from_networkx(G_hier)

for node_id, node in G_hier_cx.get_nodes():
    node['n'] = node_id_to_node_name[node_id]

G_hier_cx.set_name(d1_name+'_'+d2_name+'_systems_map')
for node_id, node in G_hier_cx.get_nodes():
    data = MP_full_results_df.loc[node['r']]
    for row, value in data.items():
        if (row.find('gene_ids')>-1) or (row=='gprofiler_name'):
            data_type = "string"
            value=str(value)
        else:
            data_type = "double"
            value = str(value) # nice cx can only accept strings as values...
```

```
            if value=='inf': # check if inf, set to -1 if so
                value='-1'

        G_hier_cx.set_node_attribute(node_id, row, value, type=data_type)

# Restore some hierarchy properties to their state before networkx conversion.
for node_id, node in G_hier_cx.get_nodes():
    for i in np.arange(len(G_hier_cx.nodeAttributes[node_id])):
        dict_temp = G_hier_cx.nodeAttributes[node_id][i]
        if dict_temp['n'] in
['CD_MemberList_Size','CD_MemberList_LogSize','HiDeF_persistence']:
            G_hier_cx.set_node_attribute(node_id, dict_temp['n'], dict_temp['v'],
type='double',overwrite=True)

# this is required so we can easily make subgraphs from systems
G_hier_cx.set_network_attribute('__CD_OriginalNetwork',
                                        values='0', type='long')

# use apply_style_from_network-- this should overwrite the existing style
netcoloc_template = ndex2.create_nice_cx_from_server('ndexbio.org',
            uuid='f338dea0-117c-11ec-9e8e-0ac135e8bacf')
G_hier_cx.apply_style_from_network(netcoloc_template)

#Upload to NDEx. Enter your ndex username and password.
# If you are new to ndex, make a new account on the website (ndexbio.org)
G_hier_cx.set_name(d1_name+'_'+d2_name+'_systems_map')
SERVER = input('NDEx server (probably ndexbio.org): ')
USERNAME = input('NDEx user name: ')
PASSWORD = getpass.getpass('NDEx password: ')
network_uuid_hier = G_hier_cx.upload_to(SERVER, USERNAME, PASSWORD)
```

21. Apply another template style to the NetColoc Systems Map for mouse variant view and export to NDEx. Select the property to be mapped to system node colors (should be one of the mouse variant phenotypes previously identified). In this style, the log odds ratio is mapped to the system node color. Systems that are not significantly enriched for the phenotype are white ($p<0.05$).
    ?Troubleshooting

```
# ------ modify this based on your project. Should be a system identified above -----
# set the property we should map to system node colors
mouse_KO_mapping_property = 'abnormal neuron morphology'

# apply a template style
G_hier_cx.set_name(d1_name+'_'+d2_name+'_systems_map_mouse_KO:'+mouse_KO_mapping_prop
erty)
# use apply_style_from_network-- this should overwrite existing style
netcoloc_template = ndex2.create_nice_cx_from_server('ndexbio.org',
            uuid='4958993c-df46-11eb-b666-0ac135e8bacf')
raw_cx_st = json.dumps(netcoloc_template.to_cx())

# replace the default template values with mouse_KO_mapping_property
```

```
updated_raw_cx = re.sub('COL=abnormal heart development:log_OR',
'COL='+mouse_KO_mapping_property+':log_OR', raw_cx_st)
updated_raw_cx = re.sub('COL=abnormal heart development:-log',
                        'COL='+mouse_KO_mapping_property+':-log', updated_raw_cx)
updated_raw_cx=json.loads(updated_raw_cx)
netcoloc_template_updated = ndex2.create_nice_cx_from_raw_cx(updated_raw_cx)
G_hier_cx.apply_style_from_network(netcoloc_template_updated)

network_uuid_hier_mouse_KO = G_hier_cx.upload_to(SERVER, USERNAME, PASSWORD)
```

22. Add genes associated with mouse variant phenotypes to the NetColoc subnetwork and export to NDEx.
    ?Troubleshooting

```
# add fields to node_df for genes in each mouse_KO phenotype of interest
MP_genes_columns = [c for c in MP_full_results_df.columns.tolist() if
c.find(':gene_ids')>-1]

# look up overlapping genes in the root node, add them to node_df
for MP in MP_genes_columns:
    focal_genes = MP_full_results_df.loc[root_node, MP].split(' ')
    node_df[MP]=0
    node_df.loc[focal_genes, MP]=1
node_df.head()
```

```
G_overlap_cx = ndex2.create_nice_cx_from_networkx(G_overlap)
G_overlap_cx.set_name(d1_name+'_'+d2_name+'_NetColoc_subgraph')
for node_id, node in G_overlap_cx.get_nodes():
    data = node_df.loc[node['n']]
    for row, value in data.items():
        if row == 'd1_seeds' or row == 'd2_seeds' or row=='sum_seeds':
            data_type = 'double'
        elif row=='d1_name' or row=='d2_name':
            data_type='string'
        else:
            data_type = 'double'
        G_overlap_cx.set_node_attribute(node_id, row, value, type=data_type)

# apply a template style (834b6ad4-d2ea-11eb-b666-0ac135e8bacf)
G_overlap_cx.apply_template('ndexbio.org','834b6ad4-d2ea-11eb-b666-0ac135e8bacf')

network_uuid_NetColoc = G_overlap_cx.upload_to(SERVER, USERNAME, PASSWORD)
```

23. Upload the cosine-similarity transformed NetColoc subnetwork to NDEx.

```
#Annotate network
print("Number of nodes in cosine similarity network:", len(G_cosSim.nodes()))
print("Number of edges in cosine similarity network:", len(G_cosSim.edges()))
G_cosSim_cx = ndex2.create_nice_cx_from_networkx(G_cosSim)
G_cosSim_cx.set_name(d1_name+'_'+d2_name+'_NetColoc_subgraph_CosSim95')
```

```
for node_id, node in G_cosSim_cx.get_nodes():
    data = node_df.loc[node['n']]
    for row, value in data.items():
        if row == 'd1_seeds' or row == 'd2_seeds' or row=='sum_seeds':
            data_type = 'double'
        elif row=='d1_name' or row=='d2_name':
            data_type='string'
        else:
            data_type = 'double'
        G_cosSim_cx.set_node_attribute(node_id, row, value, type=data_type)

# apply a template style (834b6ad4-d2ea-11eb-b666-0ac135e8bacf)
G_cosSim_cx.apply_template('ndexbio.org','2cbed84b-e5c3-11eb-b666-0ac135e8bacf')

network_uuid_NetColoc_CosSim = G_cosSim_cx.upload_to(SERVER, USERNAME, PASSWORD)
```

24. Add the 4 networks from above to a network set on NDEx.

```
# append the datestring to the network set's name to guarantee uniqueness
datestr = str(datetime.now())
networkSetURL=ndex2.client.Ndex2(host=SERVER,username=USERNAME,password=PASSWORD).cre
ate_networkset(d1_name+'-'+d2_name+' network set: '+datestr,'network set for
'+d1_name+'-'+d2_name+' NetColoc subgraph and systems map')

# parse out UUID from URL strings
networkSetUUID = networkSetURL.split('/')[-1]

networkURLs =
[network_uuid_NetColoc,network_uuid_NetColoc_CosSim,network_uuid_hier,network_uuid_hi
er_mouse_KO]
networkUUIDs = [n.split('/')[-1] for n in networkURLs]

ndex2.client.Ndex2(host=SERVER,username=USERNAME,password=PASSWORD).add_networks_to_n
etworkset(networkSetUUID,networkUUIDs)
```

Further exploration of select systems

25. Import the four networks from above to Cytoscape. Navigate to the network set on the NDEx account page (**Figure 2a**). Open each network in a new tab and click “open in Cytoscape” (**Figure 2b**).
26. Apply ‘yfiles organic’ layout to NetColoc subnetwork with network edges, and NetColoc subnetwork with cosine similarity edges.
    ?Troubleshooting
27. Apply ‘yfiles tree’ layout to the NetColoc systems map. Apply copycat layout to NetColoc systems with mouse KO style to ensure both systems map have identical layouts.
    ?Troubleshooting
28. OPTIONAL: fine-tune the layout and visualization. Some options include: a) manually adjusting positions of genes/systems so labels are legible, b) modifying color schemes, c) setting non-seed gene label transparency to 0 for large networks to improve legibility.
    ?Troubleshooting

29. If not installed already, install the ‘Community Detection’ app on the Cytoscape app. store. Analyze systems of interest. Right click on a system of interest from the NetColoc systems map. Scroll to “Apps,” then “Community Detection,” then click “View interactions for selected node.” This will bring up a prompt to select a network for which to view the interactions between genes in the selected system. Select either the NetColoc subnetwork with network edges or the NetColoc subnetwork with cosine similarity edges. A new network will be created consisting of the genes and interactions in the selected system.
30. Further analyze a system of interest by selecting genes causing a phenotype of interest when knocked out in mice. These genes are available in the node table view.

Supplemental Methods

Network propagation from identified gene sets on the selected network. Here we implement a network propagation algorithm based on random-walk with restart, which corrects for degree distribution of input seed genes. Genes of interest are used as seeds to propagate from on the selected network. The propagation algorithm is an iterative process, where at each step, heat is added to the seed genes in the network, then that heat is propagated from each gene in the network to each of its neighboring genes, moving from hot to cold. Areas of the network containing many seed genes will become very hot. If seed genes are unrelated, the entire network will remain relatively cool. We used the random walk with restart network propagation algorithm[5,6], as follows:

$$F \quad = \alpha W' F \quad + (1-\alpha) Y$$

This can be expressed iteratively as:

$$F^t = \alpha W' F^{t-1} + (1-\alpha) Y$$

where Y is the vector with length $|V|$ representing the input seed genes. The value of non-seed genes is 0, and the value of seed genes is $1/S$, where S is the total number of seed genes. W' is an adjacency matrix for all the nodes in the network, where edge weights are normalized by the degrees of the edges' end points. F^t is a vector representing the heat contained in every node in the network at time t. $\alpha \in (0,1)$ is the fraction of total heat retained at every time step.

Optimisation. The iterative form of the network propagation equation is fast for single network propagation. However, when many propagations are required, the closed-form solution is much faster. In NetColoc, thousands of propagations are required to build the expected heat distribution for the computation of the network proximity. The closed-form solution is as follows:

$$\begin{gathered} F \quad = \alpha W' F \quad + (1-\alpha) Y \;\Leftrightarrow \\ F - \alpha W' F \;=\; (1-\alpha) Y \;\Leftrightarrow \\ (I - \alpha W') F \;=\; (1-\alpha) Y \;\Leftrightarrow \\ (I - \alpha W')^{-1} (I - \alpha W') F \;=\; (I - \alpha W')^{-1} (1-\alpha) Y \;\Leftrightarrow \\ F = (I - \alpha W')^{-1} (1-\alpha) Y \end{gathered}$$

For each repetition of the network propagation algorithm on the same network only Y, the vector representing the seed gene set, varies. Therefore, $(I - \alpha W')^{-1}(1-\alpha)$ can be pre-calculated, and the results reused for every repetition:

$$F = W'' Y$$

where $W'' = (I - \alpha W')^{-1}(1-\alpha)$. W'' can be understood as a $|V| \times |V|$ matrix where each row represents the results of network propagation from a single gene. $W''Y$ can be understood as adding up the results of network propagation from each gene in the seed gene set (and normalizing by the number of seed genes). For 5000 repetitions, this optimization reduces the

algorithm's runtime from over 12 hours to 10 minutes on a typical laptop. Note that the calculation of W'' becomes a bottleneck for very dense networks, but we have tested networks with up to 40 million edges, larger than most currently available biological networks.

Adjusting for degree bias. Genes with a high degree will naturally become more "hot" than other genes[15]. To adjust for degree bias, we compare the heat of each gene to its expected distribution, given a random set of seed genes. Instead of reporting heat, here we report the z-score of the true heat compared to the expected distribution of the degree-matched null model. The expected distribution is built by running network propagation thousands of times using random seed gene sets with degree distributions matched to the original seed gene set with a binning procedure, as implemented in Guney et al[36]. All genes in the network are divided into bins based on degree, such that each bin contains a minimum number of genes (a parameter the user can set). Random seeds are then selected from those bins matching the degrees of the true seed genes. A gene-level z-score is then calculated by comparing the network propagation scores from the true seeds to the distribution of propagation scores from the random seeds. The propagation z-score represents the number of standard deviations of the true score for that gene above (positive z) or below (negative z) the mean of the null distribution. This score defines the "**network proximity**." The propagation z-score successfully corrects for degree bias, recovering the expected distribution of degrees of genes in PCNet, while the uncorrected version of network propagation has a strong bias towards high-degree nodes (**Supplementary Figure 4**).

Sampling issues with large gene sets. NetColoc works best for small to medium gene sets (between 5 and 500). This is because sampling issues occur with larger gene sets, due to the finite size of the interactome. For example, in PCNet, with seed gene sets <500, the median overlap among gene sampling pairs is < ~4%. With larger seed gene sets, this median overlap grows to 12% for 2000 genes, and >25% for 5000 genes. Furthermore, if we examine the number of gene recurrences from 100 random samplings, the most common gene appears in <20% of samples with seed gene sets < 500 genes, and the median gene appears in only 1% of samples. For larger gene sets, on the other hand, the most common gene appears in >50% of samples for seed gene sets of 5000 genes, while even the median gene appears in 25% of samples (**Supplementary Figure 1**). When randomly sampled sets become too similar to each other, the variance in the null distribution for z-score calculation becomes smaller, and so the z-scores become artificially inflated. To maintain an approximately random null distribution for the sampled sets, we recommend keeping input seed gene sets smaller than 500 genes. In cases where there are too many genes in the seed gene set, we recommend filtering the input list in some way (sorting by p-value and/or effect size, and taking the top 500, for example).

Network colocalization. The network colocalization subnetwork is built by taking the product of the two proximity vectors as follows:

$$z_{coloc} = z_1 * z_2$$

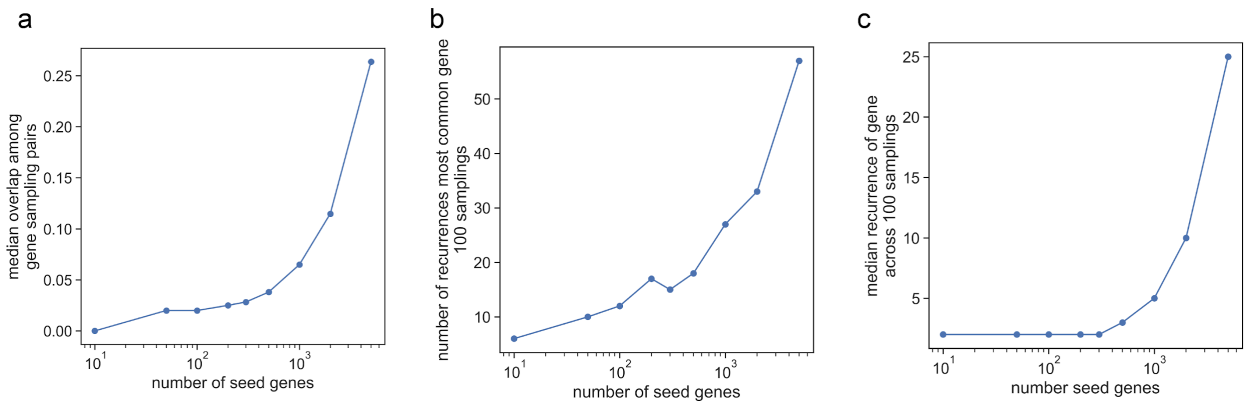
We then select genes for which z_{coloc} is greater than a threshold ($z_{coloc} > 3$ is the default, but can be set by the user), and the network proximity scores were individually larger than a nominal

threshold ($z_1 > 1.5$, and $z_2 > 1.5$ are the defaults, but can be set by the user). The subnetwork comprising the genes and interactions meeting these criteria make up the network colocalization subnetwork.

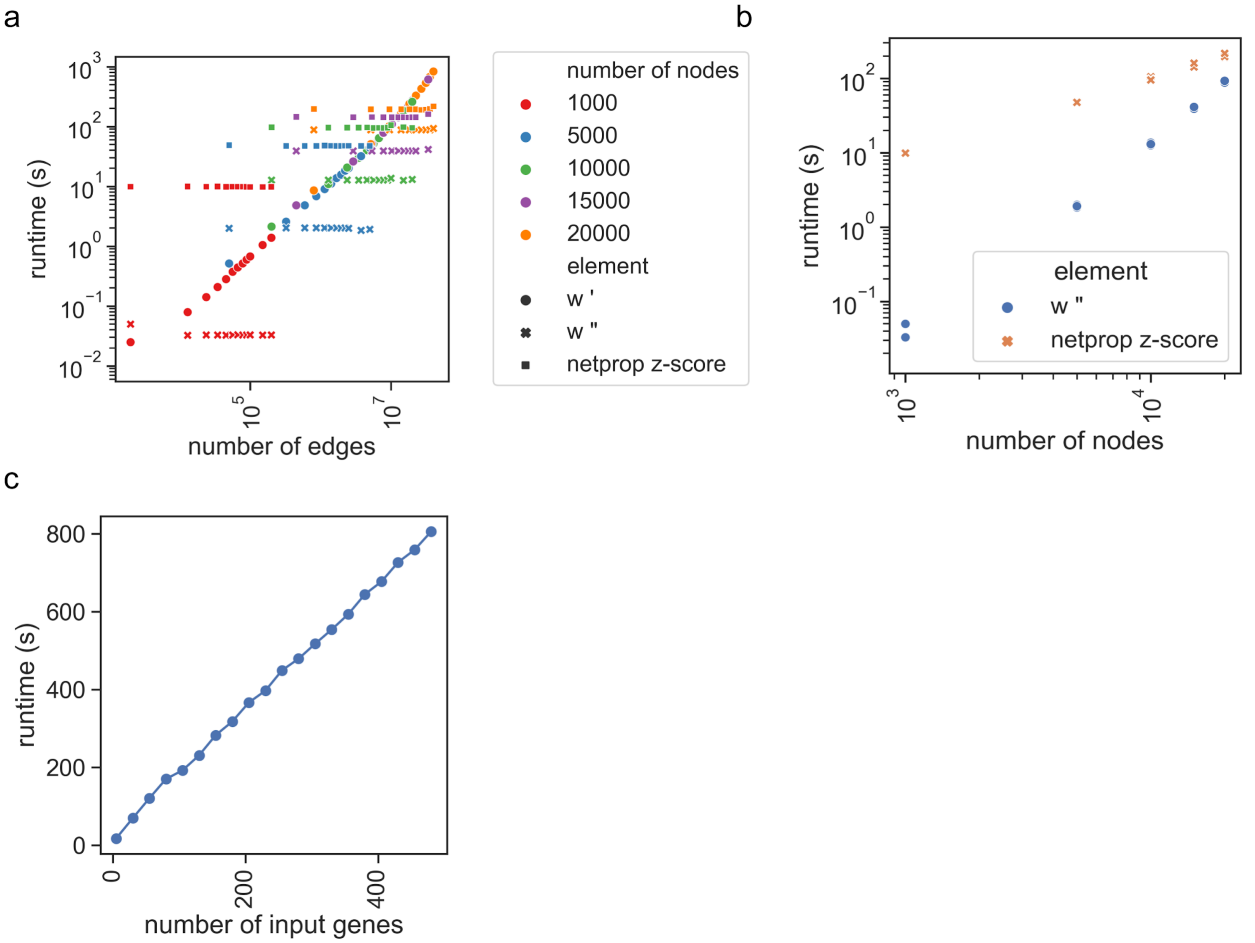
We evaluate the significance of the network colocalization by comparing the size of the network colocalization subnetwork to a null distribution. The null distribution is obtained by permuting individual z-score proximity labels 1000 times and taking the product of these shuffled z-score proximity vectors each time. Significance is assessed by comparing the observed size of the network intersection to the null distribution and computing an empirical p-value.

Mouse variant validation. Integration of the network colocalization subnetwork with mouse variant data is performed using the mammalian phenotype ontology, with data from the Mouse Genome Informatics database[16]. After parsing the ontology using DDOT[37], we identify a phenotype(s) of interest, and all genes which, when knocked out in mice, cause the phenotype. Genes causing the phenotype(s) of interest when knocked out are tested for enrichment in the full NetColoc subnetwork, using Fisher's exact test.

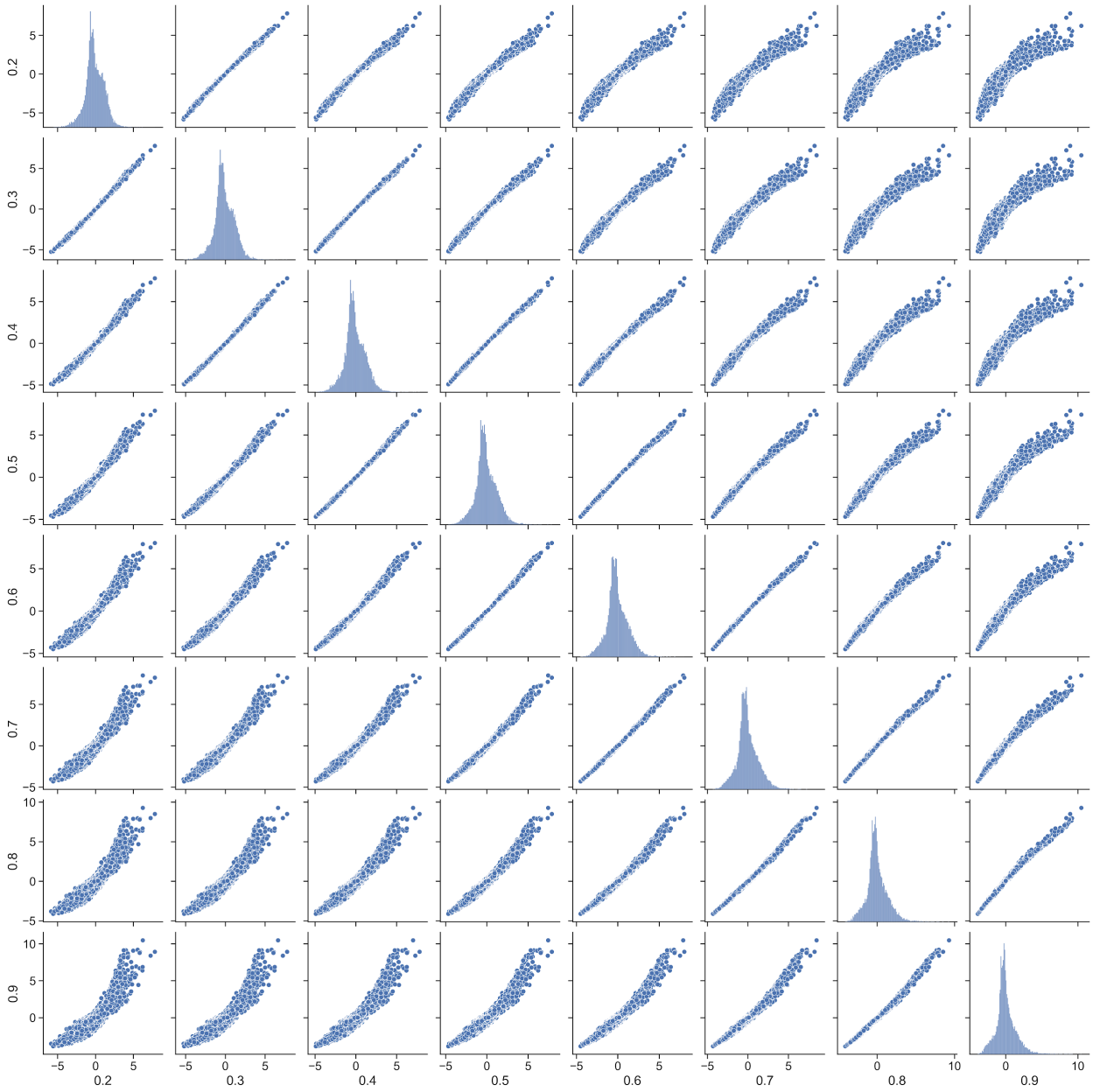
Supplemental Figures



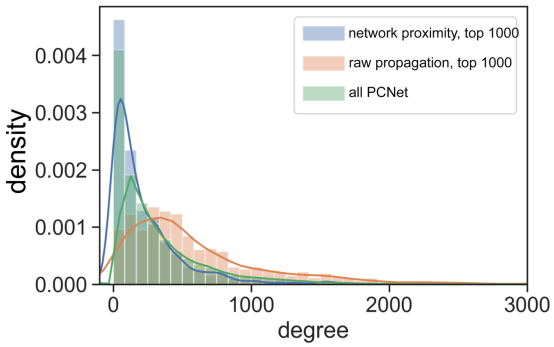
Supplementary Figure 1: Large input seed gene sets cause sampling issues. *a) Median fraction of shared genes among gene sampling pairs (degree-matched sampling). As seed gene sets become larger, sampled gene sets become more similar to each other. b) Number of recurrences of the most common gene across 100 randomly sampled sets, as a function of the size of the seed gene set. As seed gene sets become larger, each sampled gene set is more likely to contain the most common gene. c) Median number of recurrences for any gene across 100 randomly sampled sets. As seed gene sets become larger, the same genes are likely to recur in sampled sets, leading to less independence between sampling sets.*



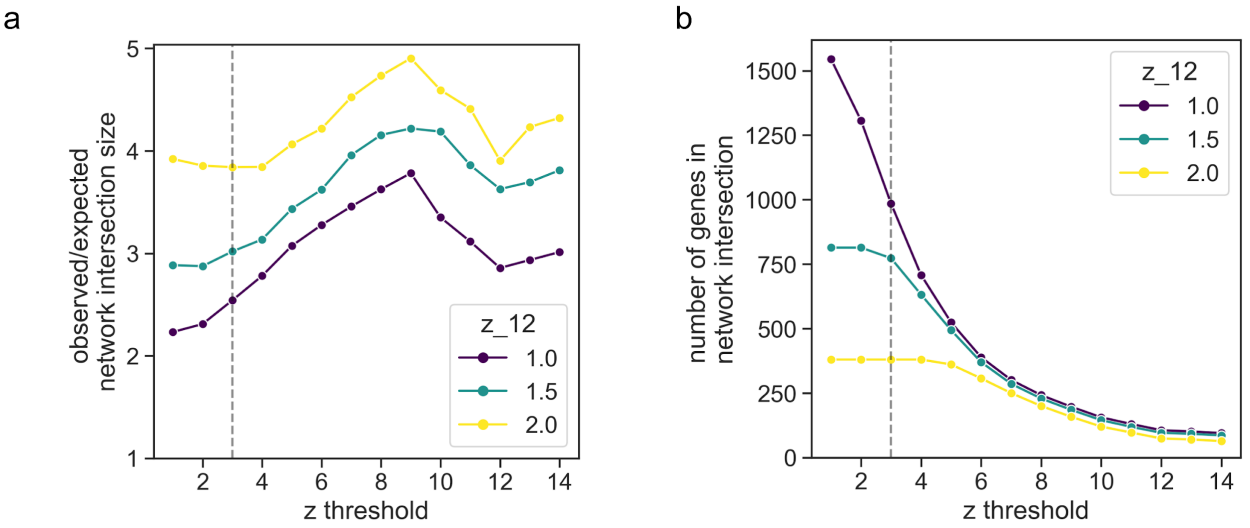
Supplementary Figure 2: Analysis of NetColoc run time. *a) Run time (y-axis) to compute matrices for propagation (w' and w''), and network proximity z-scores, as a function of number of edges (x-axis). b) Run time (y-axis) to compute w'' and network proximity z-score as a function of number of nodes (x-axis). Run time evaluated on random Erdos-Renyi graphs of different node size and edge densities. Network proximity scores initiated from input list of 100 randomly selected nodes. c) Run time (y-axis) to compute network proximity z-scores as a function of size of input gene list. Run times were calculated on a random graph with 20,000 nodes and 3 million edges, on a linux machine with 16 cores and 64 GB memory.*



Supplementary Figure 3: Network proximity scores are not strongly sensitive to the choice of diffusion parameter. *Dot plots show the correlation among network proximity z-scores using a range of diffusion parameters from 0.2 to 0.9. Diagonal elements show the distribution of propagation z-scores. Individual panel x and y axes show the network proximity z-scores.*



Supplementary Figure 4: Degree correction in network propagation recovers expected degree distribution. *The blue curve shows the distribution of degrees for the top 1000 genes ranked by network proximity score, the orange curve shows the distribution of degrees for the top 1000 genes ranked by raw network propagation score (without correction for the degree of input seed genes), and the green curve shows the distribution of degrees for all genes in PCNet. The network proximity score degree distribution closely mirrors the background degree distribution for PCNet, while the raw network propagation score is biased towards high-degree genes.*



Supplementary Figure 5: NetColoc enrichment for different choices of z-scores. *a) Ratio of observed to expected size of network intersection for a range of combined z-score thresholds (x-axis), and individual z-score thresholds (colors). b) Size of the network intersection (number of genes) or a range of combined z-score thresholds (x-axis), and individual z-score thresholds (colors).*