Supplementary Materials for “FastEnsemble: Scalable ensemble clustering on large networks”

Yasamin Tabatabaee, Eleanor Wedell, Minhyuk Park, and Tandy Warnow*

Siebel School of Computing and Data Science, University of Illinois Urbana-Champaign, Urbana, IL, USA

Contents

List of Tables

List of Figures

*warnow@illinois.edu

A Advanced mode of FastEnsemble

In its advanced mode, FastEnsemble can integrate the outputs of different clustering algorithms with arbitrary weights. Thus, FastEnsemble can take as input a set of clustering methods $M_1, M_2, \ldots, M_{n_p}$, a set of weights $w_1, w_2, \ldots, w_{n_p}$ and a set of parameters $r_1, r_2, \ldots, r_{n_p}$ as input, in addition to the threshold t. The parameter r_i represents a relevant setting or parameter for the clustering method M_i. For example, r_i can be the resolution parameter when using the Leiden algorithm for CPM-optimization. Finally, w_i defines the weight of the clustering method M_i, representing its relative influence on the final clustering output. The entries of the co-classification matrix are adjusted based on these weights, so that

$$A_{ij} = \frac{\sum_{t\in\{1,\ldots,n_p\}} w_t \mathbb{1}|v_i \text{ and } v_j \text{ are co-clustered in } C_t|}{n_p \sum w_t} \tag{A}$$

B Details of the experimental study

B.1 Simulated Datasets

B.1.1 Algorithm design datasets

We used the library NetworkX [3] with the following command to generate the LFR graphs used in the algorithm design experiment:

```
graph = nx.generators.community.LFR_benchmark_graph(n=n, tau1=3, tau2=1.5,
mu=mu, average_degree=d, min_community=10, seed=1932)
```

where the mixing parameter `mu` varies between 0.1 and 0.9, the average degree `d` varies between 5, 10 and 20, and the number `n` of nodes in the network varies between 1000 to 100,000. The parameters τ_1 and τ_2 are exponents for the degree and community size distributions respectively and c_{min} is the minimum community size. This collection of networks was used in Experiment 1.

In addition, we attempted to regenerate the LFR datasets used in Figure 2 of [4]. We were not able to re-generate these graphs using NetworkX, potentially due to the choice of parameters. Therefore we used the original implementation of LFR software in c++ available at [5] to generate these networks with the following command:

```
./binary_networks/benchmark -N 10000 -k 20 -maxk 50 -mu <mixing-parameter>
-maxc 100 -minc 10 -t1 2 -t2 3
```

Note that [6] suggest that the normal range for parameter τ_2 is 1 to 2, while [4] set this value to 3. It seems possible that this discrepancy may have resulted in problems in generating this set of networks using NetworkX. Since the parameters τ_1 and τ_2 used in our simulations for Figure 1 are in the range suggested by [6] while the parameters used in [4] are not, we consider the LFR networks we used in Experiment 1 to be preferable.

B.1.2 Synthetic datasets derived from real-world networks

For Experiments 2 and 3, we used the 34 LFR networks from [2] that were generated based on the properties of six real-world networks and their Leiden clusterings, optimizing respectively for modularity or CPM with different resolution values (refer to [2] for further information about network generation protocol). We omit those LFR graphs that had high proportion of disconnected ground-truth communities, therefore using only 27 out of 34 graphs. These are freely available in the Illinois Data Bank at https://doi.org/10.13012/B2IDB-6271968_V1.

B.1.3 Ring-of-cliques and Tree-of-cliques networks

Experiment 4 explores accuracy on ring-of-cliques networks that are formed by arranging n cliques in a ring, with each ring of size 10. We used the following command to generate these networks with the software NetworkX, where `n` and `k` determine the number of cliques and size of each clique, respectively.

```
nx.ring_of_cliques(num_cliques=n, clique_size=k)
```

To create Tree-of-Cliques networks, we used the following custom code

```
def gen_tree_of_cliques(k, n):
  cliques = [nx.complete_graph(k) for _ in range(n)]
  tree = nx.random_tree(n)
  tree_of_cliques = nx.disjoint_union_all(cliques)
  for s, d in tree.edges():
    tree_of_cliques.add_edge(s*k+k-1, d*k)
  return tree_of_cliques
```

that first generates a random tree of size `n` and `n` disjoint cliques of size `k`, and then connects the cliques according to the structure of the tree, so that for each edge `(s,d)` in the tree, the last node of clique `s` is connected to the first node of clique `d`.

B.1.4 Synthetic datasets composed of Erdős-Rényi graphs

The datasets we use in Experiment 5 are either Erdős-Rényi graphs or are composed of a combination of Erdős-Rényi (ER) graphs with an LFR graph or a ring-of-cliques.

The Erdős-Rényi graphs have 1000 nodes. We used NetworkX to generate these Erdős-Rényi graphs, with the following command, where p specifies the probability of creating an edge that affects the density of the graph:

```
graph = nx.erdos_renyi_graph(n=1000, p=p)
```

To combine these graphs with synthetic graphs with known community structure, we created an LFR graph with 1000 nodes using NetworkX with the following command

```
lfr = nx.generators.community.LFR_benchmark_graph(n=1000, tau1=3, tau2=1.5, mu
    =0.1, average_degree=9.198, min_community=45, seed=10)
```

This graph has 14 ground-truth communities with sizes [45, 47, 57, 59, 60, 61, 70, 74, 74, 87, 88, 91, 91, 96].

To attach this graph to the Erdős-Rényi graphs with various densities, we used the following commands:

```
graph = nx.erdos_renyi_graph(n=1000, p=p)
graph = nx.disjoint_union_all([graph, lfr])
graph.add_edge(0, graph.number_of_nodes() - 1)
```

These commands create an Erdős-Rényi graph using NetworkX with edge probability p and connects it to the LFR graph created before using a single edge. The ground-truth community structure for the ER-LFR graphs is assumed to be the 14 communities in the LFR graph in addition to 1000 singletons communities for the ER graph.

Similarly, we created combinations of Erdős-Rényi graphs of various densities with a Ring-of-Cliques networks with 100 cliques of size 10 each (refer to Section B.1.3 for the command for generating Ring-of-Cliques graphs).

Note that these Erdős-Rényi networks can have isolated nodes, and hence the edge list representation for these networks may suggest fewer than 1000 nodes; the full synthetic network for the Erdős-Rényi graph, however, has 1000 nodes.

B.2 Ablation study

We performed an ablation study isolating each algorithmic parameter in FastEnsemble (i.e., number of partitions, threshold, edge weighting, choice of the base clustering method) to more rigorously evaluate their individual contributions. Results are shown in Figures B, C, E, D, on the algorithm design datasets. We take the default setting for FastEnsemble and vary each parameter, one at a time, and explore the impact on accuracy (AMI, ARI, and NMI).

The four experiments explore changes to the following algorithmic parameters:

- Choice of the base clustering method that includes Leiden-mod or Leiden-CPM with four different resolution values (0.001, 0.01, 0.1, 0.5) (Figure B).
- The number np of partitions, varying between 5, 10, 50, 100 (Figure C).
- The threshold t varying between 0.6,0.7, 0.8, 0.9, 1.0 that was the optimal range in our initial Experiment 1a (Figure E).
- Whether or not to consider edge weights (Figure D).

Evaluating the impact of the base clustering method (Fig B) shows that, in most cases, Leiden optimizing modularity outperforms Leiden-CPM across all tested resolution values. However, the smallest tested resolution parameter for CPM (0.001) performs comparably to Leiden-modularity and even slightly surpasses it in one instance (at $\mu = 0.3$ in terms of ARI). The choice of resolution parameter in Leiden-CPM also affects accuracy: larger resolution values tend to reduce accuracy, while the two smallest values (0.001 and 0.01) yield similar and relatively higher accuracy (Fig B). Increasing the number of partitions (Fig C) generally improves accuracy. However, there is very little difference between the two highest values for np (50 and 100). In addition, NMI accuracy parameter is large (> 0.6), increasing np can slightly reduce accuracy in terms of AMI, but it always improves accuracy in terms of NMI (Fig C). The impact of the threshold value depends on the mixing parameter and the accuracy measure (Fig E), with better accuracy in terms of AMI and ARI resulting from large thresholds (even t=1.0 that corresponds to Strict Consensus) for networks with very low mixing parameter and our default threshold (t=0.8) providing better results for higher mixing parameters. In terms of NMI however, higher thresholds produce better results for all mixing parameters, with the three highest tested values (t=0.8,0.9,1.0) producing almost identical results (Fig E). Finally, whether edge weights are considered in the final clustering seems to have minimal impact on accuracy on these datasets (Fig. D).

B.3 Methods and Software Commands

FastEnsemble The code for FastEnsemble is available at https://github.com/ytabatabaee/fast-ensemble/blob/main/fast_ensemble.py. The following command can be used to run it, where `-t` refers to the threshold for removing weak edges and `-p` specifies the number of partitions. The flag `--noweight` specifies that the method should ignore edge weights when clustering. In our experiments, we ran FastEnsemble with Leiden-mod and Louvain for modularity clustering without the `--noweight` flag. When we used FastEnsemble with Leiden-CPM we used it with the `--noweight` flag.

```
python <git root>/fast_ensemble.py -n <edgelist> -t <threshold> -alg <algorithm> [-r <
    resolution-value>] -p <number-of-partitions> [--noweight]
```

Commit id used: 0cce8ce

FastConsensus. The FastConsensus [4] software is available at https://github.com/kaiser-dan/fastconsensus. We used the following command to run it

```
python <git root>/fast_consensus.py -f <Input network> --alg louvain -np 10 -t 0.2 -d
    0.02
```

where `-np` specifies the number of partitions, `-t` specifies the threshold for removing weak edges, `-d` is the convergence threshold and `--alg` specifies the clustering algorithm which is by default Louvain. Commit id used: 9bf993b

ECG. The software for ECG [7] is available at https://github.com/ftheberge/Ensemble-Clustering-for-Graphs/tree/master. We wrote a custom script to run it, which is available at https://github.com/ytabatabaee/fast-ensemble/blob/main/scripts/ECG.py. We used the following command to run ECG using this script:

```
python <git root>/ECG.py <Input network> <Output file>
```

Commit id used: ce21601 with igraph version 0.9.7 on Python 3.9.18

A new version of ECG implemented inside partition-igraph is available at https://pypi.org/project/partition-igraph/, and can be used with the following command inside a python script, where `final` specifies the algorithm used in the final step, that can be Leiden or Louvain.

```
import igraph as ig
import partition_igraph

ecg = g.community_ecg(ens_size=10, final='leiden', resolution=1.0)
```

Clustering Accuracy Evaluation To calculate accuracy (including NMI, AMI, ARI, FNR, FPR, precision, recall, F1-score) of clusterings computed on the synthetic networks, we used a script available at https://github.com/ytabatabaee/fast-ensemble/blob/main/scripts/clustering_accuracy.py.

This script uses the scikit-learn library to calculate NMI, AMI and ARI, and a custom code to calculate the other measures. The script for calculating accuracy can be run with the following command:

```
python <git root>/clustering_accuracy.py gt <Ground-truth membership> -p <Estimated
    partition>
```

Commit id used: 5ea5f66

On real-word networks (DBLP and Amazon Products), we used a script for NMI, ARI, and AMI accuracies, available at https://github.com/illinois-or-research-analytics/network-analysis-code/blob/main/clustering_accuracy.py. We use the scikit-learn library for ARI and graph-tool library for NMI and AMI.

```
python <git root>/clustering_accuracy.py --input-edgelist <Network edgelist> --
    groundtruth-clustering <Ground-truth membership> --estimated-clustering <Estimated
    partition> --noise-threshold 1.1 --noise-file <two column (cluster_id, noise) file>
    --output-file <clustering accuracy output file>
```

Commit id used: 0c7aa81

Mixing parameter calculation The mixing parameter of a network for a given clustering is the average, across all the nodes in the network, of the ratio between number of neighbors of the node outside its community and its total degree. Nodes that are not in any cluster (community) have a mixing parameter that depends on whether they are isolated (i.e., have no neighbors) or not. The mixing parameter for the nodes that are isolated is 0.0; those that are not isolated have mixing parameter 1.0.

On synthetic networks, we have the ground-truth clustering, and the mixing parameter was calculated using the network and that ground-truth clustering.

On the real-world networks from the SNAP repository (i.e., Amazon Products and DBLP), the ground truth communities are provided, along with a subset of the top 5000 ground-truth communities [8]. We calculate the mixing parameter based on these top 5000 communities (using the technique described above), and report those values in Table 1 in the main paper. The ground truth communities are overlapping, and to enable accuracy measurements using AMI, ARI, and NMI, we modify them to make them pairwise-disjoint by removing nodes that are in two or more communities. This results in a modified version of the ground truth communities that has much lower node coverage (i.e., many nodes are not in any community). We therefore also calculate the mixing parameter for the clustered subnetwork we derive from this process; this is done by averaging the mixing parameter for every node that appears in the clustered subnetwork (i.e., in a cluster of size at least two). Note that the mixing parameter for a node in the clustered subnetwork is based on the neighbors it has within the entire network. We report these in the main paper in the Discussion section.

The script for calculating the mixing parameter of a network/clustering pair is available at https://github.com/ytabatabaee/emulate-real-nets/blob/main/estimate_properties.py. Commit id used: 867e025

C Derivation of mixing parameter for ring-of-cliques networks

For a ring-of-cliques network with n cliques of size k, the mixing parameter with respect to the ground-truth can be calculated as

$$\hat{\mu} = \frac{1}{nk} \sum_{i \in \{1,\ldots,nk\}} \frac{d_i^{out}}{d_i^{in} + d_i^{out}} = \frac{1}{nk} \times 2n \times \frac{1}{(k-1)+1} = \frac{2}{k^2} \tag{B}$$

according to Eq 1 in main text, as d_i^{out} is zero for all except the two nodes in each clique that are connected to other cliques, and these two nodes are connected to $k-1$ nodes inside their community and one node outside, and therefore $\frac{d_i^{out}}{d_i^{in}+d_i^{out}} = \frac{1}{k}$.

D Average degrees for a collection of real-world networks

We obtained a collection of 74 networks from [1] and plotted the average degree of each network against the network size, which is shown in Figure A. The average degrees were obtained directly from the metadata of the repository.

List of datasets and their number of nodes ordered by size from smallest to largest: dnc (906); uni_email (1133); polblogs (1224); faa_routes (1226); netscience (1461); new_zealand_collab (1511); collins_yeast (1622); interactome_stelzl (1702); bible_nouns (1773); at_migrations (2115); interactome_figeys (2239); us_air_traffic (2278); drosophila_flybi (2906); fly_larva (2952); interactome_vidal (3023); openflights (3214); bitcoin_alpha (3783); fediverse (4860); power (4941); advogato (5155); bitcoin_trust (5881); jung (6120); reactome (6229); jdk (6434); elec (7115); chess (7301); sp_infectious (10972); wiki_rfa (11381); dblp_cite (12590); anybeat (12645); chicago_road (12979); foldoc (13356); inploid (14629); google (15763); marvel_universe (19251); fly_hemibrain (21739); internet_as (22963); word_assoc (23132); cora (23166); lkml_reply (26885); linux (30834); topology (34761); email_enron (36692); pgp_strong (39796); facebook_wall (45813); slashdot_threads (51083); python_dependency (58739); marker_cafe (69413); epinions_trust (75879); slashdot_zoo (79116); twitter_15m (85712); prosper (89269); wiki_link_dyn (100304); livemocha (104103); wikiconflict (116836); lastfm_aminer (136409); wiki_users (138587); wordnet (146005); douban (154908); academia_edu (200169); google_plus (211186); libimseti (220970); email_eu (265009); stanford_web (281903); dblp_coauthor_snap (317080); notre_dame_web (325729); citeseer (384054); twitter (465017); petster (623748); yahoo_ads (653260); berkstan_web (685230); myspace_aminer (853466); google_web (875713); hyves (1402673).

E Mixing parameter of real-world networks

For the DBLP network, there are 317,080 nodes with 79,535 of them in the top 5000 clusters after nodes with more than one cluster assignment have been removed. As a result, there are 169 singleton node clusters in the final disjoint ground-truth clustering. Table 1 in the main paper reports the average of mixing parameter across all 317,080 nodes in the network. The average of mixing parameters for the clustered subnetwork (non-singleton ground-truth clusters) is 0.381.

For the Amazon Products network, there are 334,863 nodes with 3179 of them in the top 5000 clusters after nodes with more than one cluster assignment have been removed. As a result, there are 52 singleton node clusters in the final disjoint ground-truth clustering. Table 1 in the main paper reports the average of mixing parameter across all 334,863 nodes in the network. The average of mixing parameters for the clustered subnetwork (non-singleton ground-truth clusters) is 0.085.

F Additional Tables

Table A: Failures to complete for FastConsensus on synthetic LFR datasets from the algorithm design experiment (left) and the [2] dataset based on modularity clusterings (right). The time limit was set to 4 hours for the algorithm design dataset and 48 hours for the [2] datasets.

n	d_{avg}	μ
1,000	10	0.9
10,000	5	-
10,000	10	0.4, 0.5, 0.8
10,000	20	0.5
100,000	10	0.1 to 0.9

	n	m
CEN-mod	3,000,000	20,821,202
open_citations-mod	3,000,000	55,128,496
cit_patents-mod	3,774,768	15,648,081

Table B: Runtime of methods on simulated modularity-based LFR networks from Experiment 2.

		runtime
LFR cit_hepph mod	FastEnsemble(default)	43s
	FastConsensus	21m 54s
	ECG	15s
	Leiden-mod	3s
LFR wiki_topcats mod	FastEnsemble(default)	7h 36m 24s
	FastConsensus	14h 34m 39s
	ECG	6h 20m 36s
	Leiden-mod	1m 38s
LFR cit_patents mod	FastEnsemble(default)	23h 1m 35s
	FastConsensus	>2d
	ECG	1d 12h 7m 37s
	Leiden-mod	2m 48s
LFR cen mod	FastEnsemble(default)	12h 8m 47s
	FastConsensus	>2d
	ECG	12h 38m 1s
	Leiden-mod	2m 31s
LFR open_citations mod	FastEnsemble(default)	1d 3h 52m 6s
	FastConsensus	>2d
	ECG	21h 58m 30s
	Leiden-mod	3m 37s

Table C: Runtime of FastEnsemble (top) and Leiden-CPM (bottom) on simulated CPM-based LFR networks used in Experiment 3. In each case, FastEnsemble is used with Leiden-CPM with the same resolution value as was used to provide parameters to the LFR software to generate the network. Here, "n.a." refers to conditions were results are not reported, either due to the LFR software not being able to generate a graph for that condition, or due to a high proportion of the ground-truth communities being disconnected.

FastEnsemble(Leiden-CPM)	0.0001	0.001	0.01	0.1	0.5
cit_hepph	22s	26s	30s	39s	56s
wiki_topcats	57m 2s	45m 21s	51m 54s	51m 19s	n.a.
cit_patents	39m 51s	34m 53s	38m 46s	45m 11s	45m 35s
CEN	1h 12m 48s	1h 7m 19s	1h 6m 7s	n.a.	n.a.
open_citations	2h 34m 58s	1h 43m 24s	1h 43m 51s	2h 39m 5s	2h 10m 57s

Leiden-CPM	0.0001	0.001	0.01	0.1	0.5
cit_hepph	20s	21s	21s	24s	23s
wiki_topcats	11m 31s	12m 5s	12 m 42s	13m 19s	n.a.
cit_patents	9m 18s	9m 6s	10m 9s	10m 14s	9m 31s
CEN	13m 54s	13m 37s	13m 25s	n.a.	n.a.
open_citations	30m 50s	29m 41s	26m 28s	30m 17s	36m 33s

G Additional Figures

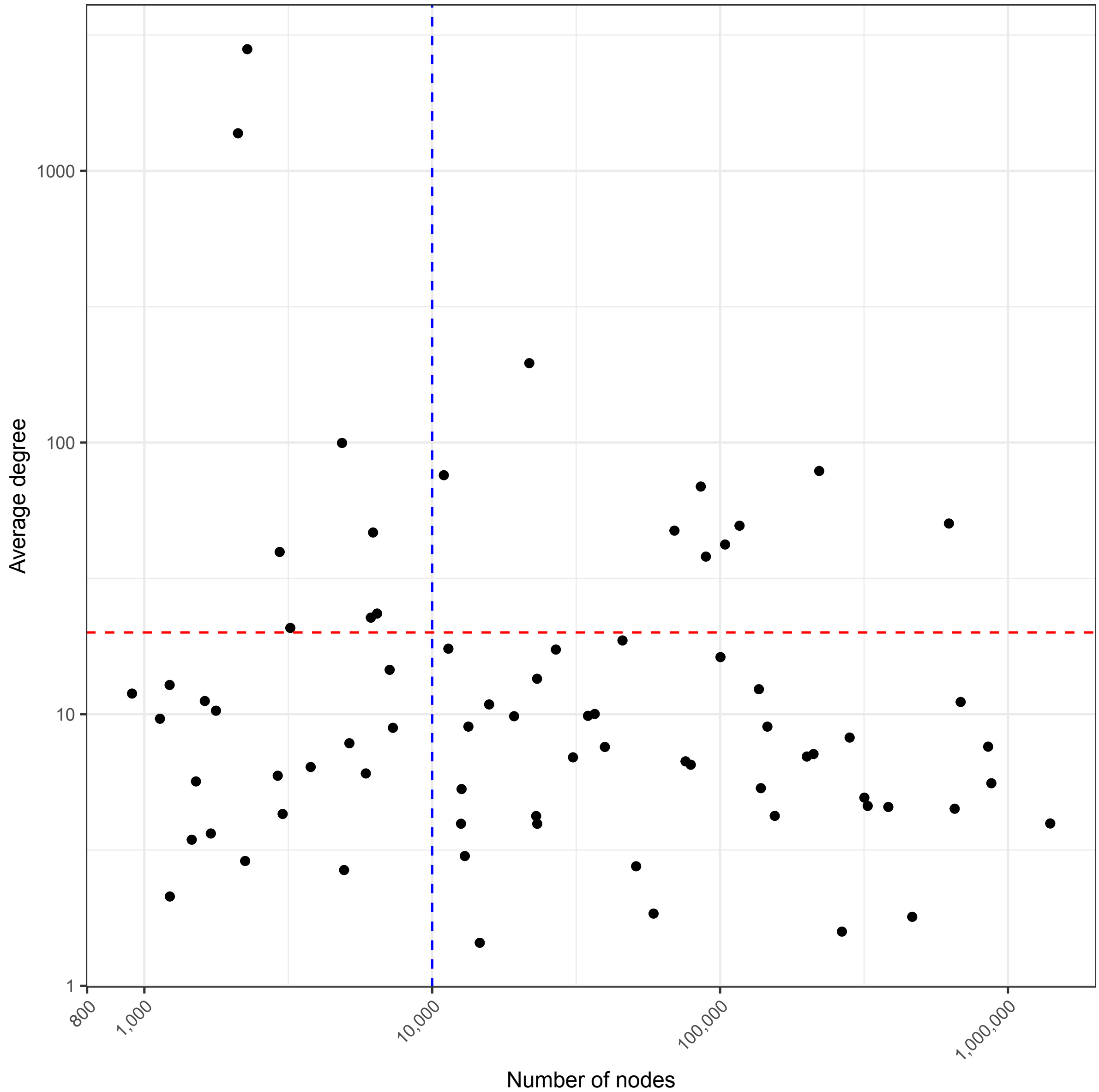


Figure A: **Average degree of real-world networks with varying sizes from the Netzschleuder network catalogue** [1]. The scatter plot shows the average degree of each real-world network ranging in size from 906 to 1,402,673 nodes. Both the x and y axes are displayed in log-scale with the blue dashed vertical line drawn at x=10,000 and red dashed horizontal line drawn at y=20. All of the networks along with their average degrees are from [1]. List of datasets ordered by size from smallest to largest: dnc; uni_email; polblogs; faa_routes; netscience; new_zealand_collab; collins_yeast; interactome_stelzl; bible_nouns; at_migrations; interactome_figeys; us_air_traffic; drosophila_flybi; fly_larva; interactome_vidal; openflights; bitcoin_alpha; fediverse; power; advogato; bitcoin_trust; jung; reactome; jdk; elec; chess; sp_infectious; wiki_rfa; dblp_cite; anybeat; chicago_road; foldoc; inploid; google; marvel_universe; fly_hemibrain; internet_as; word_assoc; cora; lkml_reply; linux; topology; email_enron; pgp_strong; facebook_wall; slashdot_threads; python_dependency; marker_cafe; epinions_trust; slashdot_zoo; twitter_15m; prosper; wiki_link_dyn; livemocha; wikiconflict; lastfm_aminer; wiki_users; wordnet; douban; academia_edu; google_plus; libimseti; email_eu; stanford_web; dblp_coauthor_snap; notre_dame_web; citeseer; twitter; petster; yahoo_ads; berkstan_web; myspace_aminer; google_web; hyves.

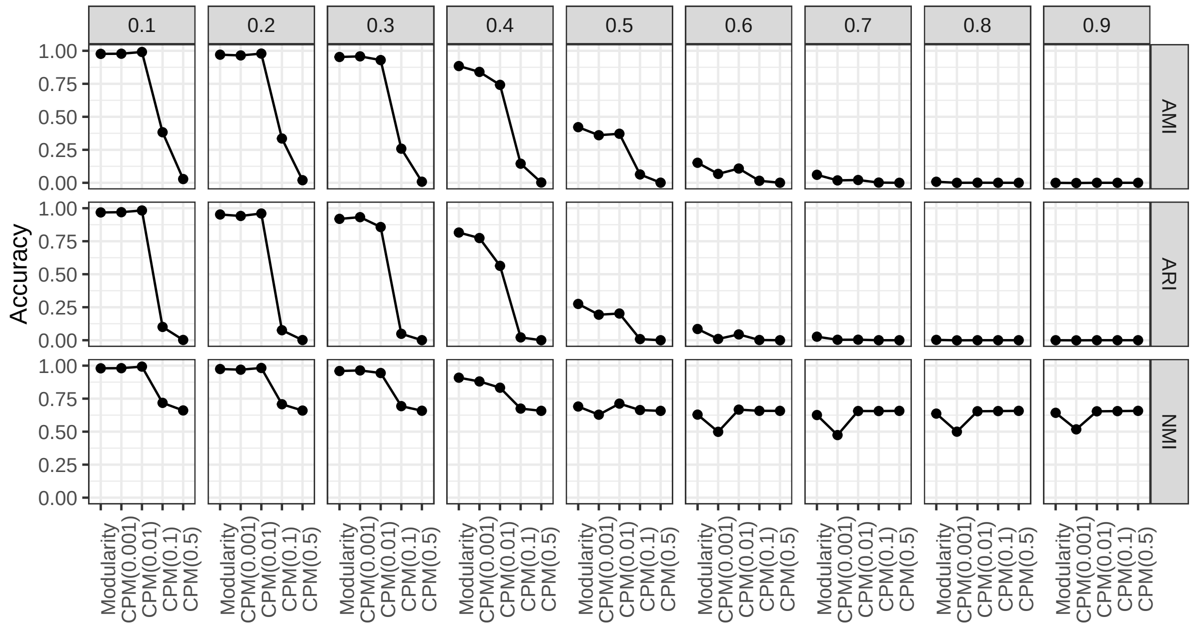


Figure B: **Ablation experiment varying the base clustering method in FastEnsemble.** The methods compared are FastEnsemble with Leiden-modularity and Leiden-CPM with resolution values 0.001, 0.01, 0.1, 0.5. Accuracy is measured in terms of AMI, ARI and NMI. The datasets used are the algorithm design dataset from Experiment 1 for conditions with the default average degree (10) and network size (10,000); values on the x-axis are the mixing parameter values for generating the synthetic networks.

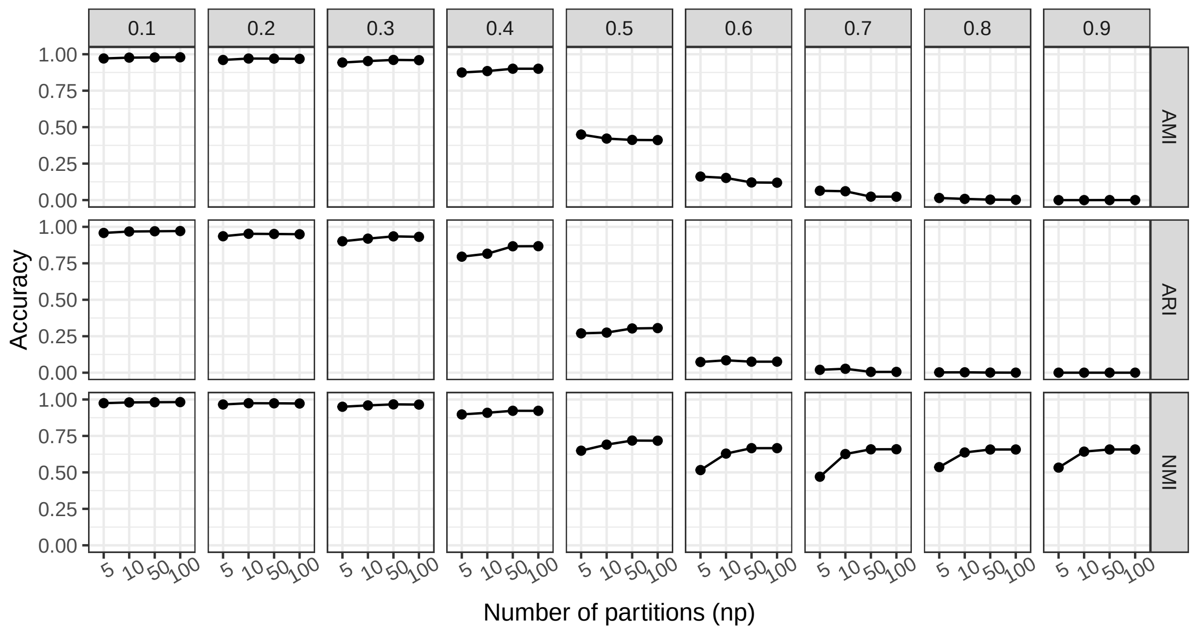


Figure C: **Ablation experiment varying the number of partitions in FastEnsemble.** Varying the number of partitions (np) in FastEnsemble between 5, 10, 50 and 100. Accuracy is measured in terms of AMI, ARI and NMI. The datasets used are the algorithm design dataset from Experiment 1 for conditions with the default average degree (10) and network size (10,000); values on the x-axis are the mixing parameter values for generating the synthetic networks.

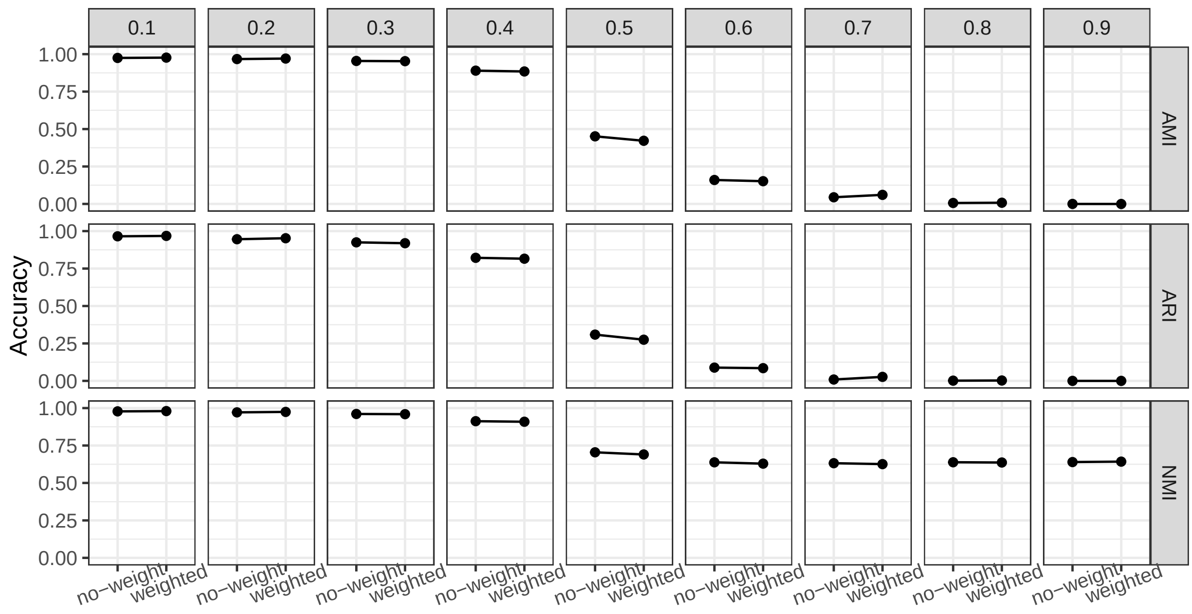


Figure D: **Ablation experiment varying whether weights are considered in FastEnsemble.** The clustering method used is Leiden-modularity, with and without considering edge weighting. Accuracy is measured in terms of AMI, ARI and NMI. The datasets used are the algorithm design dataset from Experiment 1 for conditions with the default average degree (10) and network size (10,000); values on the x-axis are the mixing parameter values for generating the synthetic networks.

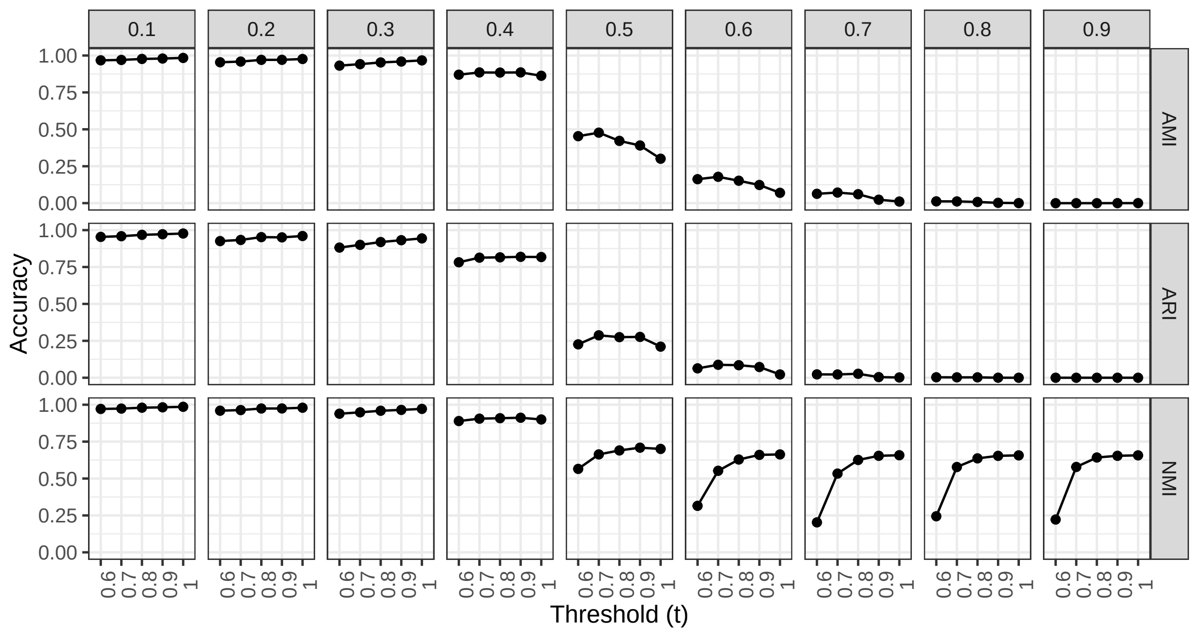


Figure E: **Ablation experiment varying the threshold in FastEnsemble.** Varying the threshold in FastEnsemble between 0.6, 0.7, 0.8, 0.9 and 1.0 (which is equivalent to Strict Consensus). The clustering method used is Leiden-modularity. Accuracy is measured in terms of AMI, ARI and NMI. The datasets used are the algorithm design dataset from Experiment 1 for conditions with the default average degree (10) and network size (10,000); values on the x-axis are the mixing parameter values for generating the synthetic networks.

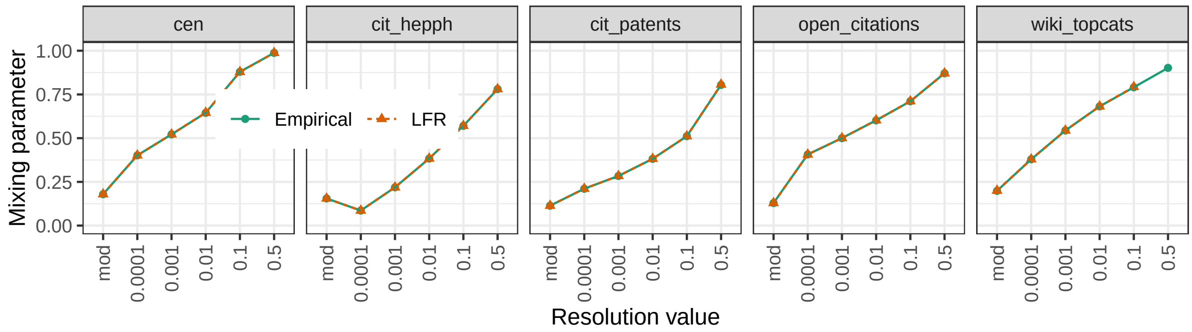


Figure F: **Mixing parameters for LFR networks based on clustered real-world networks for Experiments 2 and 3, and their corresponding real-world networks**. Each condition on the x-axis corresponds to a *different* LFR network, generated based on Leiden-modularity or Leiden-CPM with that specific resolution parameter. The mixing parameter results for the real-world (empirical) networks are calculated using the specified Leiden clustering, and for the synthetic network using the LFR ground-truth community structure (the green and orange lines respectively). No results are shown for wiki_topcats network with resolution value $r = 0.5$ due to LFR failing to generate networks for that setting. Note that the LFR network based on Leiden-modularity clustering has the smallest mixing parameter, and that the mixing parameters for the LFR networks based on Leiden-CPM clusterings increase with the resolution value. Note also that the mixing parameters for the real-world and corresponding LFR networks are nearly identical. (Figure modified from Figure 5.2 in [9].)

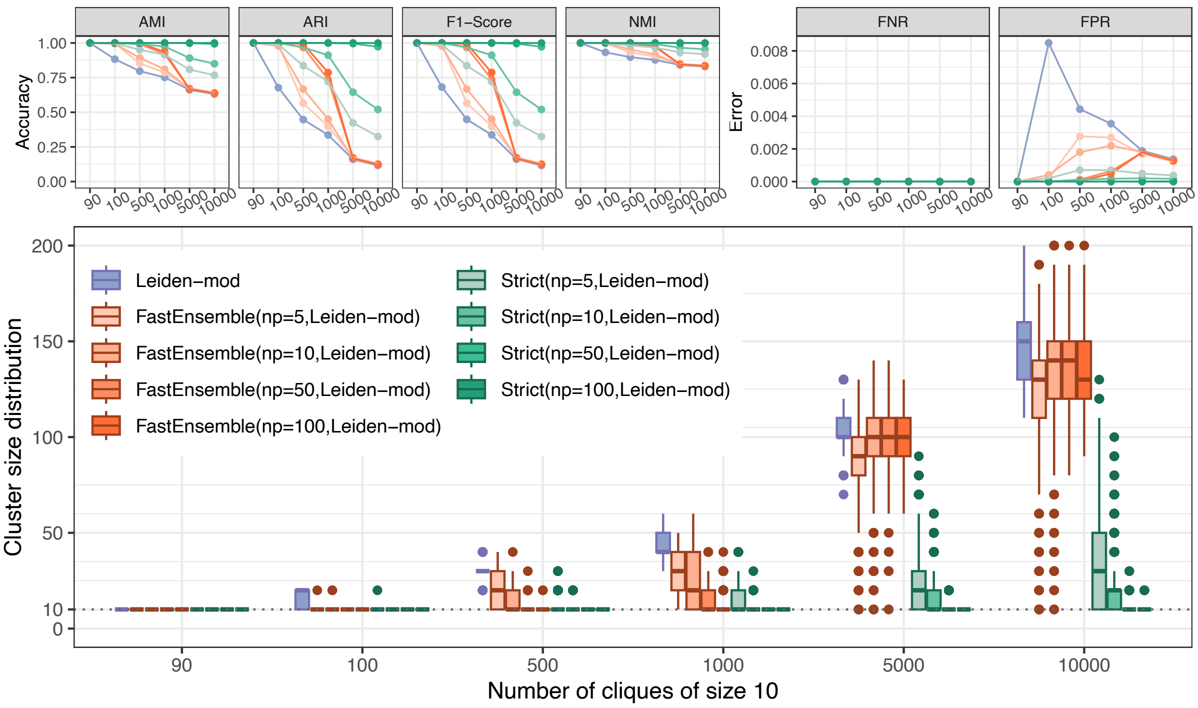


Figure G: **Impact of varying** np**, the number of partitions, on the accuracy of FastEnsemble and Strict Consensus for modularity-based clustering on ring-of-cliques networks.** The conditions correspond to Experiment 4 in the paper. Each ring-of-cliques networks connects n cliques of size 10 in a ring. The methods compared are FastEnsemble and Strict Consensus, each with four different numbers np of partitions, as well as Leiden-mod. Top left: Accuracy (AMI, ARI, NMI, F1-score) as a function of n. Top right: Error metrics (FNR and FPR) as a function of n. Bottom: Cluster size distribution as a function of n (the dotted line indicates the true distribution). There is one synthetic network for each number of cliques, hence no error bars are shown.

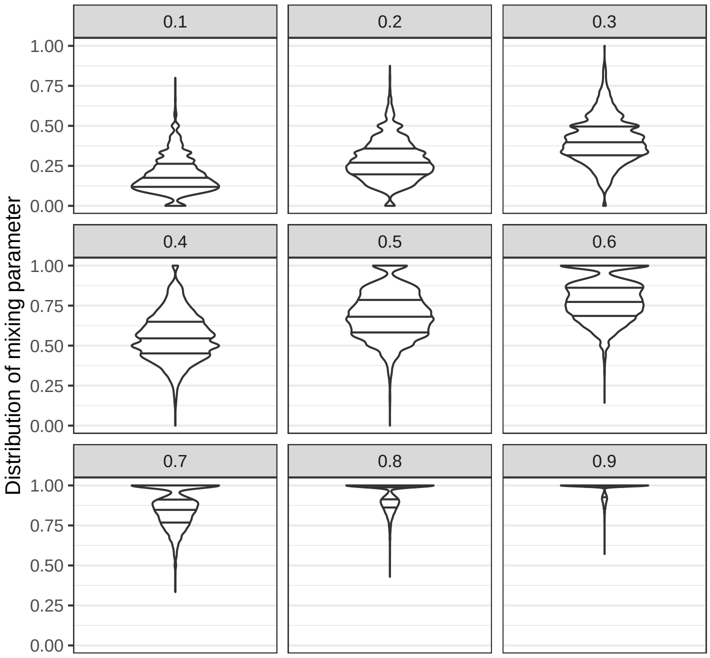


Figure H: **Distribution of the per-node mixing parameters for the networks used in Experiment 1.** Each panel corresponds to a *model* mixing parameter used to generate the LFR network and the y-axis shows the *estimated* mixing parameter. Mixing parameters are calculated with respect to the LFR ground-truth community structure. The 25%, 50% and 75% quantiles are specified with black lines on the violin plots. The conditions have the default average degree (10) and network size (10,000).

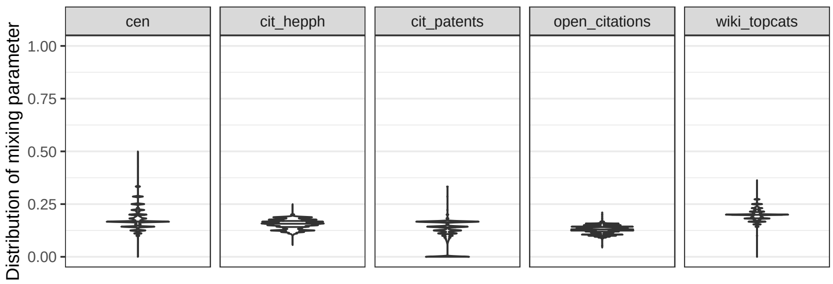


Figure I: **Distribution of the per-node mixing parameters for the modularity-based LFR networks from [2] used in Experiment 2.** The LFR networks are generated based on a Leiden-mod clustering of a corresponding empirical network (shown on the panels). Mixing parameters are calculated with respect to the LFR ground-truth community structure.

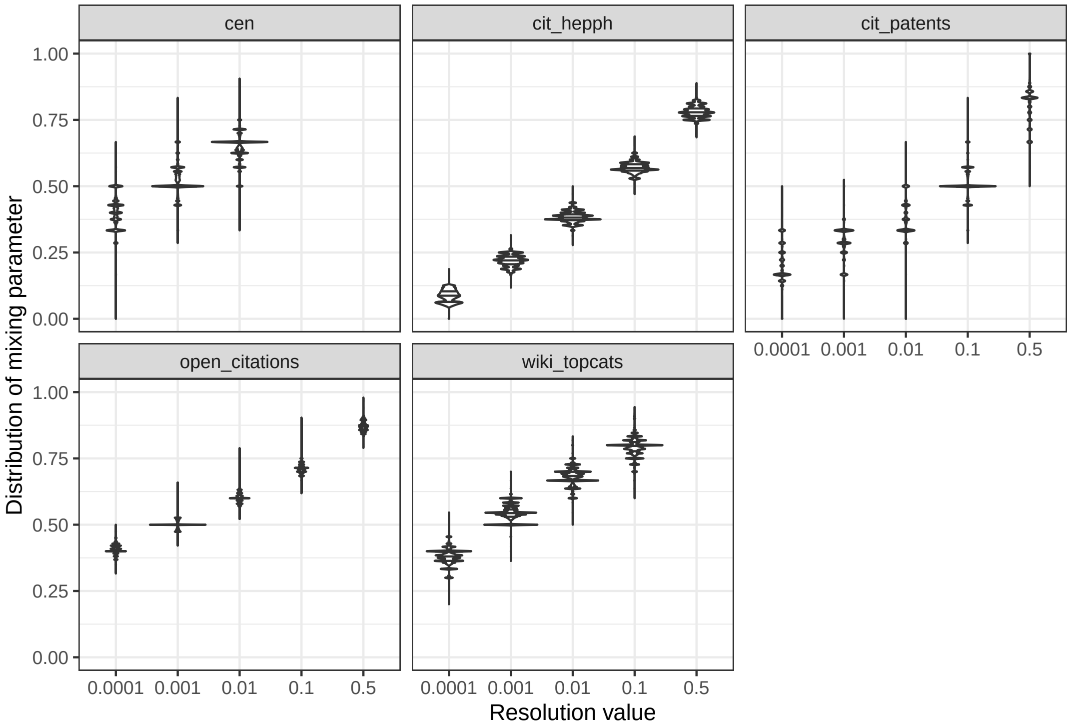


Figure J: **Distribution of the per-node mixing parameters for networks used in Experiment 3.** Each condition on the x-axis corresponds to a *different* LFR network corresponding to an empirical network (shown on the panels), generated based on Leiden-CPM with that specific resolution parameter; these networks are taken from [2]. Mixing parameters are calculated with respect to the LFR ground-truth community structure. Results are not shown for three conditions: LFR graphs with a large fraction of disconnected ground truth clusters (the two CEN networks) or when the LFR software failed to create a network for the provided parameters (the wiki_topcats network).

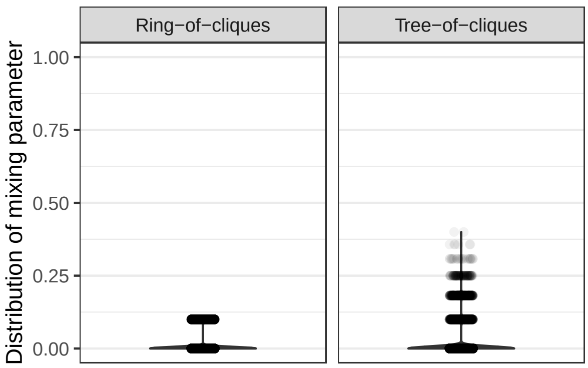
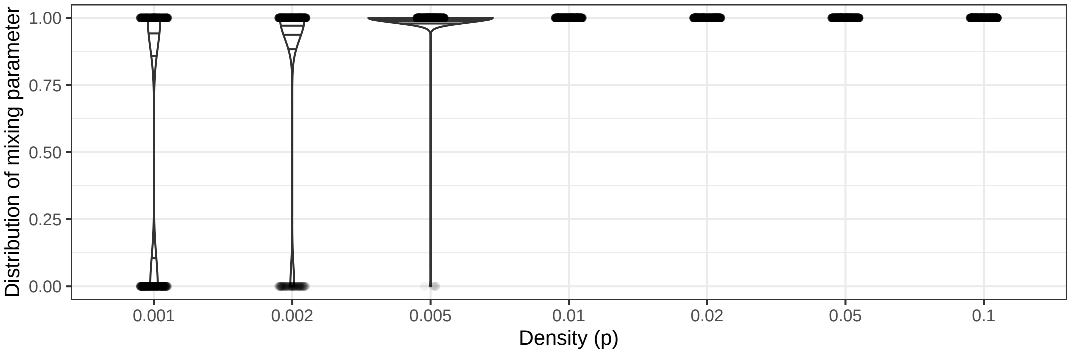
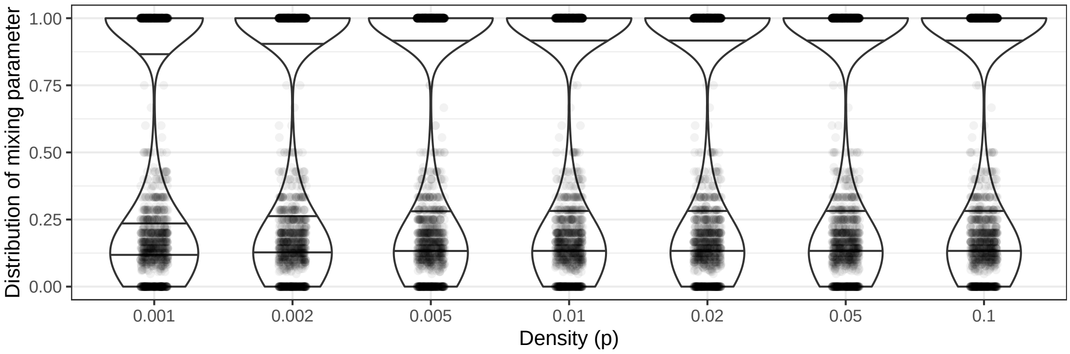


Figure K: **Distribution of the per-node mixing parameters for the Ring-of-cliques and Tree-of-cliques networks used in Experiment 4.** Only one example of each network is shown, since the distributions are independent of the network size for Ring-of-cliques and Tree-of-cliques networks. The ring and tree of cliques have 1000 cliques of size 10.

A) Erdős-Réyni



B) Erdős-Réyni+LFR



C) Erdős-Réyni+Ring-of-cliques

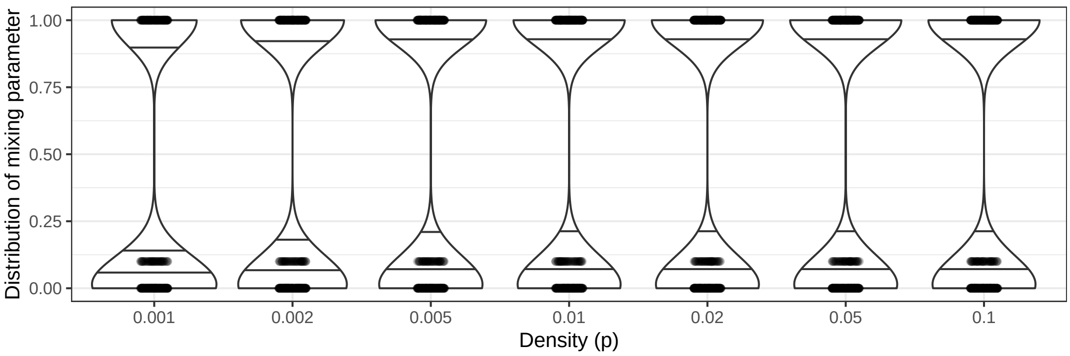


Figure L: **Distribution of the per-node mixing parameters for networks used in Experiment 5.** A) Erdős-Rényi graphs with size 1000 nodes and various densities. In figures B and C, the network is produced by combining an Erdős-Réyni network with 1000 nodes to another network of 1000 nodes with very low mixing parameter. B) Distribution for Erdős-Rényi+LFR networks. C) Distribution for Erdős-Rényi combined with a ring-of-cliques network, where each clique is of size 10. The x-axis specifies the density of the Erdős-Rényi graph. Mixing parameters are calculated with respect to the ground-truth community structure (no clusters of size greater than 1 in the Erdős-Réyni graphs). The 25%, 50% and 75% quantiles are specified with black lines on the violin plots. The black points show the actual values of mixing parameters per-node that is variable for Erdős-Rényi+LFR graphs, and take one of the three values of 0, 0.1 or 1 for Erdős-Rényi+Ring-of-cliques (see Sec. B).

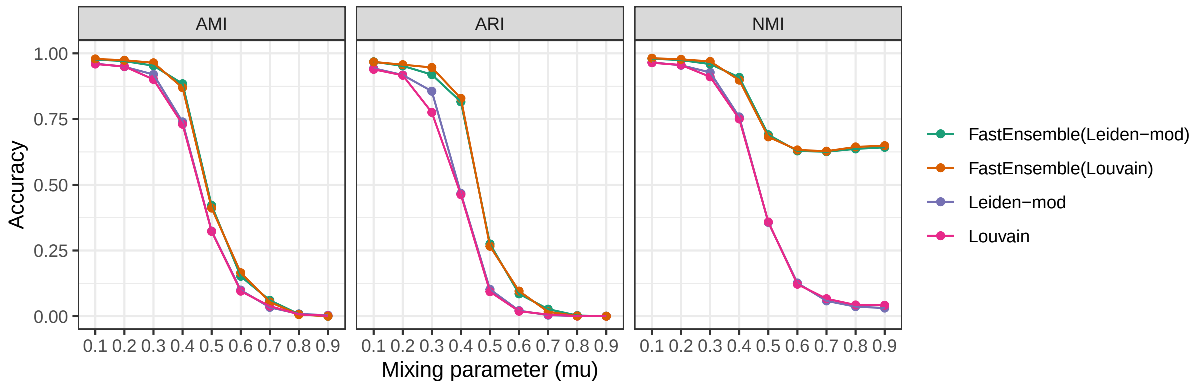


Figure M: **Comparison of Louvain and Leiden-mod on Algorithm Design datasets.** The methods compared are Leiden, Louvain, and FastEnsemble used with these two methods. The plots show accuracy of clustering methods on the algorithm design dataset from Experiment 1 for conditions with the default average degree (10) and network size (10,000); mixing parameters on the x-axis are the parameter values for generating the synthetic networks. Note that Leiden-mod and Louvain have nearly identical accuracy under all conditions, FastEnsemble used with Leiden-mod is nearly identical to FastEnsemble used with Louvain, and FastEnsemble used with Louvain or Leiden-mod are more accurate than Louvain or Leiden-mod.

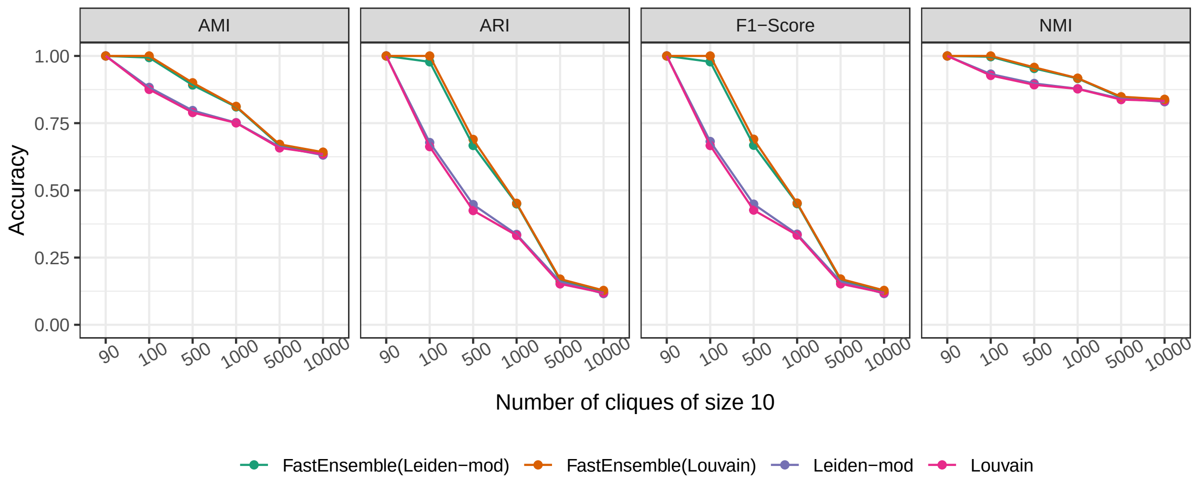


Figure N: **Comparison of Louvain and Leiden-mod on Ring-of-Cliques networks of various sizes**. The methods compared are Louvain and Leiden-mod, as well as FastEnsemble used with these methods. Each network consists of a varying number of cliques of size 10 arranged in a ring, where the total number of cliques ranges from 90 to 10,000 (specified on the x-axis). Accuracy is shown using NMI, ARI and F1-score. Note that Leiden-mod and Louvain have nearly identical accuracy under all conditions, FastEnsemble used with Leiden-mod is nearly identical to FastEnsemble used with Louvain, and FastEnsemble used with Louvain or Leiden-mod is at least as accurate as Louvain or Leiden-mod.

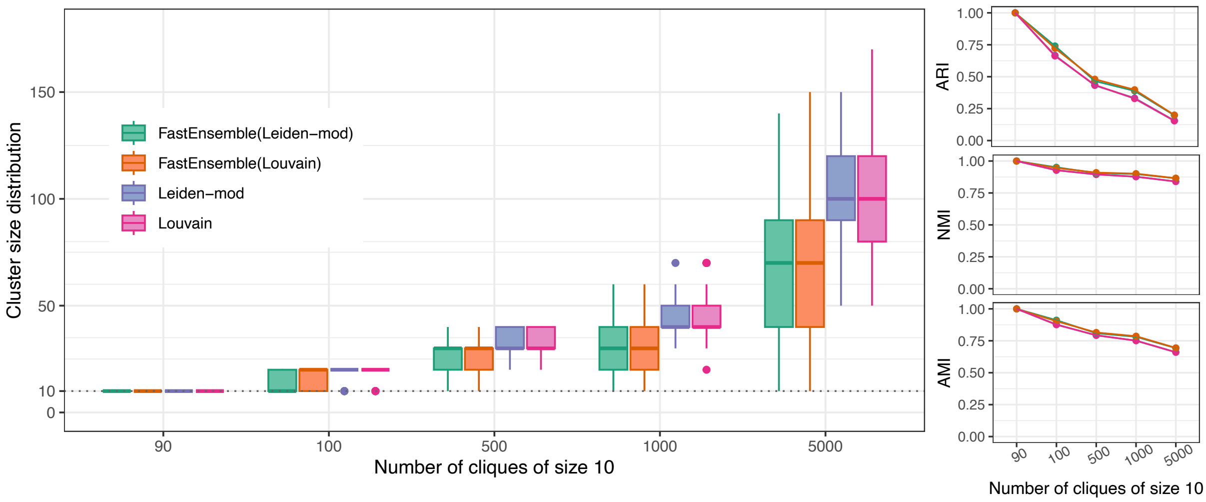


Figure O: **Comparison of Louvain and Leiden-mod on Tree-of-Cliques networks.** The methods compared are Louvain and Leiden-mod, as well as FastEnsemble used with each of these. Each network consists of a varying number of cliques of size 10 arranged according to the structure of a random tree, where the total number of cliques ranges from 90 to 5,000 (specified on the x-axis). Leiden-mod and Louvain have nearly identical accuracy under all conditions, FastEnsemble used with Leiden-mod is nearly identical to FastEnsemble used with Louvain, and FastEnsemble used with Louvain or Leiden-mod is at least as accurate as Louvain or Leiden-mod.

References

[1] Peixoto TP. The Netzschleuder network catalogue and repository; 2020. https://networks.skewed.de.

[2] Park M, Tabatabaee Y, Ramavarapu V, Liu B, Pailodi VK, Ramachandran R, et al. Well-connectedness and community detection. PLOS Complex Systems. 2024;1(3):e0000009.

[3] Hagberg A, Swart P, S Chult D. Exploring network structure, dynamics, and function using NetworkX. 7th Python in Science Conference (SciPy2008). 2008; p. 11–15.

[4] Tandon A, Albeshri A, Thayananthan V, Alhalabi W, Fortunato S. Fast consensus clustering in complex networks. Physical Review E. 2019;99(4):042301.

[5] Fortunato S. Resources; 2025. Available from: https://www.santofortunato.net/resources.

[6] Lancichinetti A, Fortunato S, Radicchi F. Benchmark graphs for testing community detection algorithms. Physical Review E. 2008;78(4):046110.

[7] Poulin V, Théberge F. Ensemble clustering for graphs: comparisons and applications. Applied Network Science. 2019;4(1):51.

[8] Leskovec J, Krevl A. SNAP Datasets: Stanford Large Network Dataset Collection; 2014.

[9] Tabatabaee Y. Improving the accuracy of community detection methods using Connectivity Modifier [Master's Thesis]. University of Illinois at Urbana-Champaign; 2023.