The Graph Lottery Ticket Hypothesis: Finding Sparse, Informative Graph Structure

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph learning methods help utilize implicit relationships among data items, thereby reducing training label requirements and improving task performance. However, determining the optimal graph structure for a particular learning task remains a challenging research problem.

In this work, we introduce the Graph Lottery Ticket (GLT) Hypothesis – that there is an extremely sparse backbone for every graph, and that graph learning algorithms attain comparable performance when trained on that subgraph as on the full graph. We identify and systematically study 8 key metrics of interest that directly influence the performance of graph learning algorithms. Subsequently, we define the notion of a "winning ticket" for graph structure – an extremely sparse subset of edges that can deliver a robust approximation of the entire graph's performance. We propose a straightforward and efficient algorithm for finding these GLTs in arbitrary graphs. Empirically, we observe that performance of different graph learning algorithms can be matched or even exceeded on graphs with the average degree as low as 5.

1 Introduction

Graph data naturally arises in many domains, including social networks, interactions on the Web, and in many biological applications. Building graphs directly from data proves useful for massive-scale data analysis; for instance, graphs can be clustered in near-linear time [18].

In recent years, graph machine learning has become a dominant paradigm in analysis of network data. The performance of many graph learning algorithms is heavily dependent on the structure of data in terms of the graph curvature [53, 49], intrinsic dimensionality [54], or many other metrics [42]. A natural compulsion is to rewire graphs to optimize such metrics. However, adding or rewiring edges may hallucinate connections that could never exist – violating the natural graph structure.

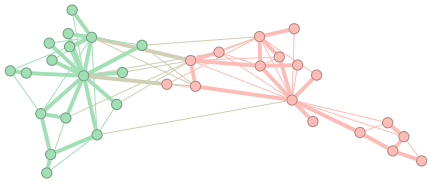

Figure 1: The Graph Lottery Hypothesis postulates that there is a sparse substructure (a *winning ticket*) present in all graphs which captures its utility for graph learning tasks. The winning ticket of the Karate club graph [59] in bold.

In this paper, we investigate the general problem of finding sparse subgraphs well-suited for graph learning – *graph lottery tickets*. Unlike most existing work, we focus on finding substructures *already present* in data, just like the "lottery tickets" in deep neural networks parameters [21]. We briefly formalize this notion as follows:

Hypothesis 1 (**Graph Lottery Ticket Hypothesis**)
Any graph contains a sparse subset of edges that—when trained on that subset only—any graph learning algorithm can match the performance of the original graph.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

We summarize our key contributions as follows:

- We formulate the Graph Lottery Ticket (GLT) hypothesis that implies the existence of an extremely sparse backbone for every graph for which graph learning algorithms attain comparable performance as on the full graph.
- We propose a straightforward yet efficient algorithm to recover "winning tickets" – extremely sparse subgraphs which still preserve task performance.
- Our experimental results illustrate our method's effectiveness. The winning tickets (sparse networks) we find match the performance for three graph learning algorithms, but with much lower average degree (≈ 5).

2 Preliminaries and Related Work

This section reviews previous attempts to optimize the structure of graphs for graph learning tasks including approaches that change the graph structure implicitly. Before diving into the related work, Section 2.1 establishes basic notation to be used throughout the paper.

2.1 Preliminaries

A graph is a pair $G = (V, E)$ with n vertices $V = (v_1, \cdots, v_n)$, $|V| = n$, and edges $E \subseteq V \times V, |E| = m$, represented by an adjacency matrix $\mathbf{A}$ for which $\mathbf{A}_{ij} = 1$ if $e_{ij} \in E$[1] is an edge between nodes i and j, otherwise $\mathbf{A}_{ij} = 0$. We denote the neighborhood set of the node u as $N(u) = v : (u, v) \in E$. Then, $\#_\Delta(i, j) = N(i) \cap N(j)$ denotes the set of triangles with the edge (i, j). For generality and simplicity of notation, we assume undirected and unweighted graphs, however, content of the paper can be easily generalized to the weighted and directed cases.

The degree of a node is defined as $d_i = |N(i)|$, and the degree matrix $\mathbf{D}$ is the diagonal matrix with node degrees $\mathbf{D}_{ii} = d_i$. The combinatorial (unnormalized) Laplacian matrix of a graph is defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$. Its normalized counterpart $\tilde{\mathbf{L}}$ is defined as $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$, where $\mathbf{I}$ is the identity matrix. We use $(\lambda_1, \cdots, \lambda_n)$ to denote the ordered set of eigenvalues of graph Laplacians and $(\mu_1, \cdots, \mu_n)$ – of graph adjacency.

2.2 Graph Sparsifiers and Spanners

Graph sparsifier is a sparse subgraph that preserves particular properties of the original graph. For instance, the surprising fact that ε-approximate cut sparsifier with $\tilde{\mathcal{O}}(n/\varepsilon^2)$ edges can be constructed in $\tilde{\mathcal{O}}(m)$ time was first established in [30, 6]. That notion was strengthened [52] to *spectral sparsifiers* – a graph $\tilde{G}$ is called a spectral sparsifier of G if

$$(1 - \varepsilon)x^\top \mathbf{L}_{\tilde{G}} x \leq x^\top \mathbf{L}_G x \leq (1 + \varepsilon)x^\top \mathbf{L}_{\tilde{G}} x$$

for all $x \in \mathbb{R}^V$. Cut sparsifiers are only required to satisfy these inequalities for all $x \in \{0, 1\}^V$. The factors hidden in $\tilde{\mathcal{O}}$ are, however, large. Good sparsifiers, e.g. [51, 5], are computationally expensive, limiting their practicality. More scalable solutions, e.g. [22], are restricted to cut sparsification and do not guarantee graph connectivity, which is crucial for many graph learning algorithms.

Spanners [44] provide a combinatorial view to sparsification. Instead of preserving algebraic properties of linear systems, spanners preserve the distances in graphs with multiplicative (t-spanners) or additive ($+\beta$-spanners) distortion. [1] propose to find t-spanners via a generalization of the classical greedy minimum spanning tree algorithm due to [34]. [9] proposes a way to sparsify near-cliques during graph construction process. The modified graph is provably a 2-hop spanner of the original, however, the number of spurious added edges can be of size of the graph itself. In general, it is unclear how graph distances translate to the performance of graph learning algorithms.

2.3 Graph Rewiring

Graph rewiring approaches aim to optimize the structure of a given graph via changing, adding, or deleting edges. A heuristic edge-swap algorithm was proposed in [10] to optimize multiple spectral

[1] For readability purposes we use "node i" instead of v_i here and further, wherever appropriate.

graph robustness measures (which we review in Section 3) with updates computed using matrix perturbation theory. The same strategy is used in [31] with an even more crude update approximation for improving the algebraic connectivity, leading to improvements in learning graph neural networks. In a similar vein, [53] propose a greedy rewiring algorithm for optimizing the structure of a graph for a modified definition of augmented Forman curvature. A different optimization metric was offered by [4]: they flip edges that minimize the number of triangles in a graph. These methods introduces spurious edges to the graph and keep the total number of edges approximately the same. Similarly, [13] proposes to sparsify a graph iteratively with training a GNN model. In contrast, this works finds extremely sparse subgraphs *without* spurious edges and in a model-agnostic fashion.

Contrapositively, [24] propose to augment the edges of the graph with extra edges derived from the diffusion process from the original graph. This approach densifies the graph to an extreme degree, sometimes adding hundred times more edges than in the original graph.

2.4 Implicit Graph Rewiring

Many graph learning methods implicitly modify the graph to achieve scalability linear in terms of the number of nodes. A common approach for scaling up GNN training to large graphs is to sample rooted subgraphs from each node [27, 12]. While graph that were implicitly sampled during GNN training have constant degree in theory, the upper bound, assuming parameters from [27], is 2500 neighbors per node, which significantly densifies the graph. In another vein, [3] propose to rewire the subgraphs *during* GNN training to optimize the connectivity of these sampled subgraphs. This approach densifies local subgraphs and is not applicable to general graph learning algorithms.

The same is true for sampling in the process of graph embedding. DeepWalk [45] samples long random walks from each node, and further densifies the implicit graph by running a long-range window An example more amenable for analysis is the sampling process of personalized PageRank-based embedding methods, e.g. [55]. Even with approximate computation [2], PPR values of the neighborhood nodes are $\mathcal{O}(\alpha(1-\alpha)) \gg 0$, meaning the graph is densified to an extreme degree.

3 What is a Good Graph Structure?

Structural graph properties have an outsized impact on the performance of graph learning algorithms, however, to our knowledge, there is no systematic study of the phenomenon. This section covers that from two different perspectives on graph structure: spectral expansion properties and local edge curvature. Through these two lenses we try to answer the question in the section title—what does make graph structure good?

3.1 Spectral Properties

Laplacian systems are at the heart of many graph machine learning, including label propagation [60], clustering [38], and more. Condition number $\kappa(\mathbf{A}) = \frac{\lambda_n}{\lambda_1}$ bounds the convergence rate of iterative algorithms for solving linear equations in $\mathbf{A}$. Since graph Laplacians are singular, the convergence can be instead measured in terms of the finite condition number $\kappa_f = \lambda_n/\lambda_2$. From a signal propagation perspective, λ_2 is related to the worst-case mixing of a random walk over G.

Algebraic connectivity, the second eigenvalue of the graph Laplacian, is ubiquitous due to its relation to vertex connectivity. For instance, $\lambda_2 \geq \frac{4}{nD}$, where D is graph's diameter, but the most exciting appearance of λ_2 is arguably in the Cheeger constant $h(G)$ of a graph, which is the lowest-density cut of the graph normalized by cut size. Algebraic connectivity can be used to bound the Cheeger constant [14]: $\frac{\lambda_2}{2} \leq h(G) \leq \sqrt{2\lambda_2}$.

Over-smoothing in GNNs happens with the rate of $\mathcal{O}((s\lambda_2)^L)$, where s is the largest singular value of node features and L is the number of GNN layers [41, 8]. While high oversmoothing does not sound very desirable, [31] showed that relational GCNs are *flexible* in how much the smooth the graph, in the range of $[0, \lambda_2]$, as measured by the Dirichlet energy of the GCN layer. Therefore, having large algebraic connectivity should be considered advantageous from graph neural network perspective.

High λ_2 implies that a graph can not be well embedded in $\mathbb{R}$ [25]. For higher-dimensional Euclidean embeddings, [54] empirically studies the reconstruction ability with respect to the spectral dimen-

sionality of graphs. Instead of computing the spectral dimensionality directly, they estimate the graph Laplacian eigenvalue growth rate. While it may be easier to embed graphs with small λ_2, we are interested in the most informative subgraphs of a given graph. Therefore, evidence from both GNNs and graph embedding points to positive effects for maximizing λ_2, which we study in Section 5.3.

Graph robustness studies [15] introduced two additional spectral measures. Spectral radius—the largest eigenvalue of the adjacency matrix—controls the speed of various dynamic processes defined on graphs, for instance, the spread of contagious viruses. Total number of spanning trees can be thought of as the total number of ways information can be transmitted in the network. Due to the matrix-tree theorem, it can be efficiently approximated as a product of the eigenvalues of the graph Laplacian. We use both spectral radius and the number of spanning trees in our experimental study.

3.2 Curvature

Graph curvature [20, 40] adapts the notion of "flatness" from manifolds to graphs. Near-cliques tend to have large positive curvature, planar grids have zero curvature, and trees have negative curvature. Forman curvature is the most computationally efficient version that is also easier to analyze combinatorially. There are multiple definitions of Forman curvature, we introduce the one due to [46], since it was shown that augmented Forman curvature is tightly correlated with definition due to [40].

Definition 3.1. For any edge (i, j) the augmented Forman Ricci curvature is given by

$$F^{\#}(i,j) = 4 - d_i - d_j + 3\gamma|\#_{\Delta}(i,j)|, \qquad \gamma > 0.$$

An exciting recent development [17] connects the notion of the *effective resistance* to curvature of graphs. Effective resistance is defined through the Moore-Penrose pseudoinverse of the graph Laplacian $\mathbf{L}^{\dagger}$ as $\omega(i,j) = (e_i - e_j)^{\top}\mathbf{L}^{\dagger}(e_i - e_j)$.

Definition 3.2. For a node i, the link resistance curvature is given by $\rho_i = 1 - \frac{1}{2}\sum_{j\in N(i)}\omega(i,j)$.

All notions of curvature have intimate connections to the number of triangles. Effective resistance of an edge is bounded by the number of triangles containing this edge: $\omega(i,j) \leq \frac{2}{\#_{\Delta}(i,j)+2}$. [49] proves that it is impossible to faithfully embed triangle-rich graphs in the Euclidean space[2]. This provides evidence against having too many triangles in the graph for faithful embedding.

There is evidence [53] that large negative curvature leads to over-squashing of the gradients in graph neural networks. However, negative negative curvature is not strictly bad for GNNs – [16] shows how propagating the information alongside the edges of a random expander graph with small negative curvature empirically improves performance of GNNs.

These results in graph curvature motivate us to include a scalable approximation [56] to the total number of triangles in a graph and its total effective resistance $R = \sum_{i,j\in E}\omega(i,j) = n\sum_i \lambda_i^{-1}$ as metrics in experiments in Section 5.3. Additionally, we include a bound [29] on the Ollivier's notion of curvature by the means of local graph clustering coefficient of [57]. In total, we will experimentally study three metrics related to graph curvature.

4 Finding Winning Graph Lottery Tickets

As we can see from the previous section, there is no single metric dictating performance of graph learning algorithms. Therefore, a one-size-fits-all algorithm that can produce graph lottery tickets that optimize all the metrics simultaneously does not exist. Instead, this section presents two straightforward yet effective approaches to finding lottery ticket structure in general graphs in a scalable and effective way, which approximately optimize the metrics discussed above.

We want to stress that our formulation of GLT does not require knowledge of *which* graph learning algorithm will be run on the graph nor any extra information such as node features or labels. Additionally, being algorithm-agnostic implies that a successful GLT search algorithm must preserve graph connectivity, since most graph learning algorithms rely on that notion.

These requirements naturally leads us to the notion of *spanning trees*. Specifically, we propose to take a union of k random spanning trees as our GLT construction. This approach was used to construct

[2] [11] shows how nonlinear embedding models are able to circumvent this restriction.

Algorithm 1 kTree$(G, \bar{m})$

Input: Graph G, target number of edges $\bar{m}$.
Output: GLT of G.
$GLT \leftarrow (V, \emptyset)$
while $|E_{GLT}| \leq \bar{m}$ **do**
  $T \leftarrow$ RandomTree(G).
  if $|E_{GLT}| \leq \bar{m} - n + 1$ **then**
    $GLT \leftarrow GLT \cap T$
  else
    $GLT \leftarrow$ RandomSelect$(T, \bar{m} - |E_{GLT}|)$
  end if
end while
Output GLT.

Algorithm 2 1Tree$(G, \bar{m})$

Input: Graph G, target number of edges $\bar{m}$.
Output: GLT of G.
$GLT \leftarrow$ RandomTree(G)
$GLT \leftarrow$ RandomSelect$(E_G, \bar{m} - n + 1)$
Output GLT.

expander graphs and spectral sparsifiers in [26]. Algorithms 1 presents the version that we use in our experiments. Given an edge budget $\bar{m}$, we iteratively combine random spanning trees of G to form the GLT graph. We also experimentally study a more bare-bone version, 1Tree, which constructs a *single* random spanning tree and adds random edges of G to that tree (cf. Algorithm 2).

There are many exciting connections of random spanning trees to various properties of graphs, mainly through the algebraic lens of the matrix-tree theorem. One of the most interesting connections is to the notion of the effective resistance: the probability of the edge being included in a random spanning tree is in fact equal to its effective resistance.

Theorem 4.1 ([26])**.** *The union of two random spanning trees of the complete graph on n vertices has constant vertex expansion with probability $1 - o(1)$.*

Random trees were recently used as graph sparsifiers [23]. They show that a slightly advanced version (with extra edge reweighting step) of the Algorithm 1 produces a spectral sparsifier in the sense of Equation 2.2. Constructing a random spanning tree takes near-linear $\mathcal{O}(m^{1+o(1)})$ time in terms of the number of edges m, due to a recent algorithm due to [47]. Therefore, both kTree and 1Tree are almost linear in the number of the edges of the input graph. In the next section we show that in addition to attractive computational properties, both kTree and 1Tree provide significant improvements on graph learning metrics studied in Section 3.

5 Experiments

We present a wide range of experiments on real and synthetic graphs using (arguably) the three most popular graph learning algorithms:

- Louvain graph clustering [7] greedily partitions the input graph hierarchically optimizing the modularity of the graph.
- DeepWalk graph embedding [45] trains a shallow neural network on a dataset of short random walks to extract node embeddings in $\mathbb{R}^d$.
- Graph convolutional networks [32] uses the graph structure to propagate information for making graph-informed predictions.

In each experiment, we sparsify a a graph and run analyses on the sparse graph backbone. Since some of our metrics depend on the total number of edges in the graph, we use a fixed number of edges corresponding to a target average node degree from the range $[1.1, 10]$. Some graphs in our studies have an average node degree of less that 10 naturally, in this case, we stop at that number.

5.1 Baselines

We evaluate against two state-of-the-art baselines:

- **Spectral radius** [10, 31]: each edge is weighted as the gradient the spectral radius of the adjacency matrix of a graph.
- **Edge significance** [19] computes statistical edge significance for every edge. We note that for undirected and unweighted graphs this weighting strategy is equivalent to computing the contribution of an edge to the modularity metric [37].

Most graph learning algorithms require input graph to be connected, moreover, some of the metrics introduced in Section 3 are sensitive to the number of connected components in graphs. Because of that, we slightly modify competing methods to first find a minimum spanning tree of a graph with respect to the weights produced by respective baseline, and then greedily add remaining edges. For graph learning algorithms that are not sensitive to disconnected components we additionally report results of a completely **random** baseline. We do not report graph-level statistics for that strategy, as many of the metrics are not defined for disconnected graphs.

5.2 Datasets

We evaluate the proposed search method on a wide selection of 7 natural graphs, 3 graphs constructed from the data, and a set of synthetic stochastic blockmodel (SBM) graphs [39]. We provide a brief description of real-world datasets in the Appendix A.1. We randomize the train and test splits using the strategy of [50] and pick 20 nodes per class as a training set, and leave all other nodes for testing.

SBM is a generative graph model which divides graph vertices into k classes, and then places edges between two vertices i and j with probability p_{ij} derived from the assignments. Specifically, each vertex i is given a class $y_i \in \{1, \ldots, k\}$, and an edge (i, j) is added with probability $\mathbf{P}_{y_i y_j}$, where $\mathbf{P}$ is a symmetric $k \times k$ matrix containing the between/within-community edge probabilities. We set $\mathbf{P}_{y_i y_j} = q$ if $i = j$ and to p otherwise. In this simple setup, p/q is the signal-to-noise ratio that measures the strength of the assortativity of a graph. For our graph statistics study, we vary $n \in [1000, 10000]$ and set $k = 10$, $p/q = 5$, and $\bar{d} = 100$. We observe no significant performance fluctuations when varying other parameters.

5.3 Graph Robustness Measures

We evaluate five graph robustness measures from [10] as well as two versions of the clustering coefficients of the graph. For measures that require knowledge of all eigenvalues, we approximate the quantity via stochastic Lanczos quadrature method [56] with 100 starting vectors and 10 iterations. We provide a brief description of the measures, indicating whether a particular measure is ideally maximized (↑) or minimized (↓):

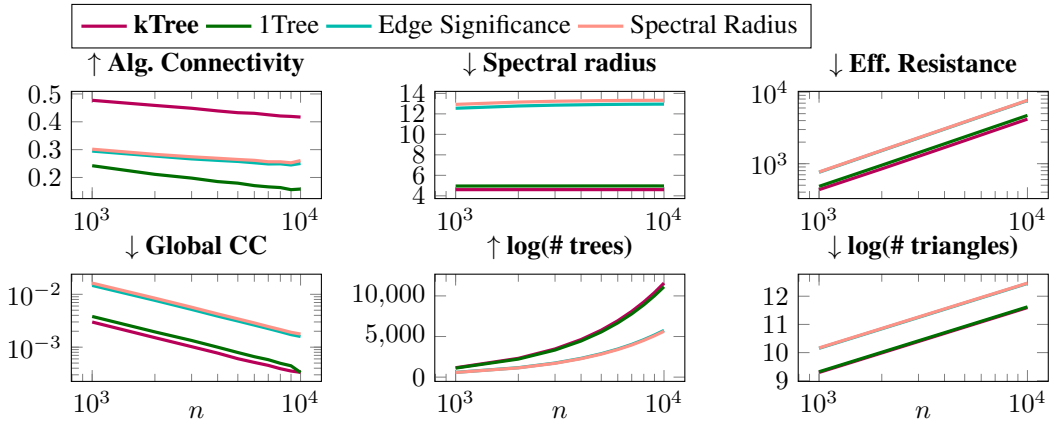

Figure 2: Graph statistics measured on stochastic blockmodel graphs, averaged acros 1000 graphs with p/q ratio of 5, sparsified to average degree of 2.

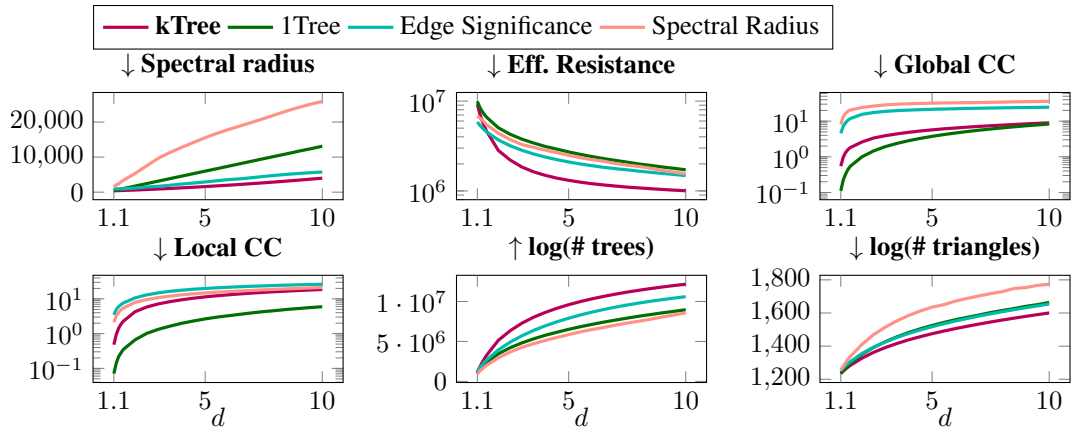

Figure 3: Statistics measured on the ε-nearest-neighbor graph constructed from the MNIST dataset.

- $\uparrow$ **Algebraic connectivity** is the smallest eigenvalue of the combinatorial graph Laplacian.
- $\downarrow$ **Spectral Radius** defined as the largest eigenvalue of the adjacency matrix of a graph.
- $\downarrow$ **Effective resistance** computed as $R = n \sum_i \frac{1}{\lambda_i}$.
- $\uparrow$ **Number of trees** computed[3] as $\log S = \sum_i \lambda_i$.
- $\downarrow$ **Number of triangles** computed as $\#\Delta = \frac{1}{6} \sum_i \mu_i$.
- $\downarrow$ **Global clustering coefficient** [36] is defined as $\mathrm{Tr}\, \mathbf{A}^3 / \sum_{i \neq j} \mathbf{A}^2_{ij}$.
- $\downarrow$ **Average local clustering coefficient** [57] is defined as $c_i = \sum_{j \in N_i} \sum_{k \in N(i)} |e_{jk}| / d_i(d_i - 1)$. We average c_i across all nodes in the graph.

We present results on the synthetic SBM graphs on Figure 2. Interestingly, the only metric with a critical difference between the kTree and 1Tree strategy is the algebraic connectivity of a graph. Overall, we can observe a big difference between tree-based and greedy selection strategies, sometimes in the orders of magnitude better for random tree-based methods.

We present results on an exemplar MNIST graph on Figure 3. Figures for all other datasets can be found in Appendix. There, we observe dramatic differences between approaches in terms of all of the metrics considered. For real graphs, we do not report λ_2 because of numerical instabilities of finding it precisely in case when it is very close to 0. Note how the differences in terms of the tree number are in logathmic terms, meaning kTree is better than the competitors by several orders of magnitude. Compared to synthetic graphs, we observe stark contrast between different methods.

5.4 Graph Clustering

We now discuss the performance of the graph clustering algorithms on sparsified graphs. For each graph, we cluster it using the Louvain method [7] for community detection. Figure 4 reports the normalized mutual information between the clustering of the sparsified graph and ground-truth node labels on both natural and nearest neighbor graphs.

We observe that unweighted random tree-based methods produce significantly better results than their weighted counterparts regardless for both edge significance and spectral radius-based strategies[4]. kTree is significantly better than 1Tree strategy on Amazon-PC, OGB-ArXiv, and MNIST datasets. We can attribute that to the overall larger correlation of the label information to the ground-truth labels. There is no case where it is losing to 1Tree. In stark comparison, both weighting strategies of [10, 31] and [19] significantly underperform on all graphs we considered, with most degradation occurring in the very sparse regime. This trend will continue in the other experiments, perhaps with a less severe trend: in general, we observe significant degradation of quality of all graph learning algorithms when using these sparsification techniques. We do not report results of the completely random baseline, as it produces many disconnected components which get assigned a separate cluster, and NMI is ill-defined for these solutions.

[3] We omit the $log(n)$ normalization factor.

[4] One might assume that there is an error in the weight calculation; however we have checked this thoroughly.

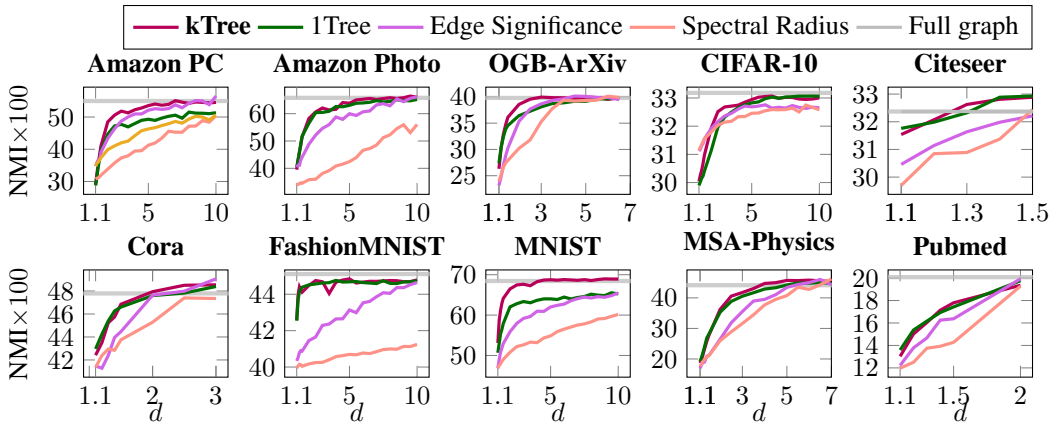

Figure 4: Clustering results on 10 real-world datasets. We vary the target average degree d and report the normalized mutual information (multiplied by 100 for convenience) with respect to the ground-truth labels in each dataset. Random baseline is not present in this study due to the fact that disconnected components produce disconnected components that make NMI overly optimistic.

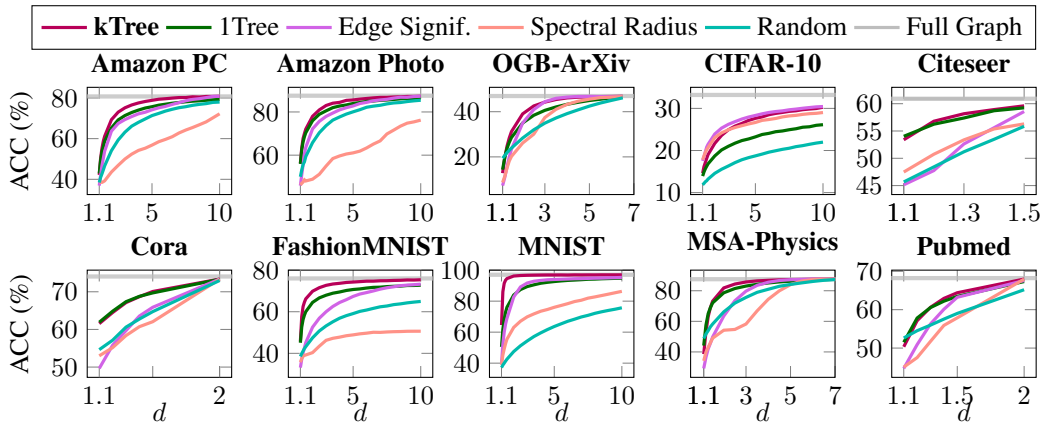

Figure 5: Graph embedding performance on 10 real-world datasets. We vary the target average degree d and report classification accuracy with respect to the ground-truth labels.

Averaged across all datasets, the budget required for the best sparsification method to match the performance of graph clustering on the whole dataset is only 2–5 edges per node. The only exception is Pubmed, where the graph structure seems to be very efficient, and all sparsification algorithms bring the performance down.

5.5 Graph Embedding

We now discuss the performance of graph embedding on sparsified graphs. For each graph, we train a graph embedding [45] with parameters from the original paper (dimensionality 128, 80 walks per node of length 80, window size 10). Then, we train a logistic regression model using scikit-learn [43] with default parameters to predict the node labels.

Figure 5 presents the results on 8 most informative datasets. We observe that random tree-based methods are superior yet again, however, this time there is a noticeable difference in performance between kTree and 1Tree on almost all datasets. We attribute that to the fact that DeepWalk algorithm performs aggressive smoothing of the input graph, so explicit decorrelation of the edges in the construction of kTree is more beneficial in this case.

Spectral radius-based weighting strategy is again performing the worst. However, in the case of graph embedding, we can compare it to the random baseline: in 3 cases, it is significantly worse, in 2 it is better and in 3 more they are tied. In this experiment, we can finally observe the extreme gains we can get by preserving the connectivity structure of graphs: the difference between the random baseline and kTree on MNIST dataset at its peak is more than 50% in terms of accuracy!

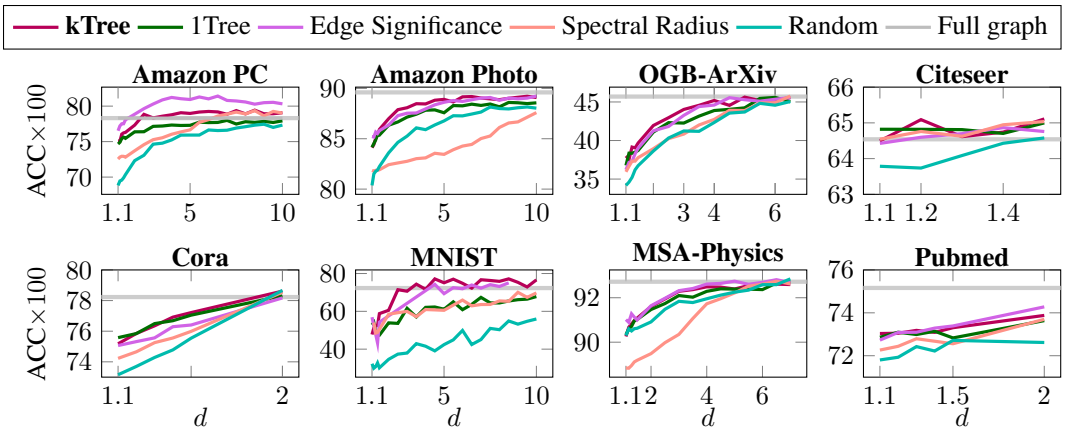

Figure 6: GNN training results on 8 real-world datasets. We vary the target average degree d and report the normalized mutual information (multiplied by 100 for convenience) with respect to the ground-truth labels in each dataset.

5.6 Graph Neural Networks

We proceed with evaluating the performance of graph neural networks on sparsified graphs. To unify the experimental setting across the For each graph, we train a basic Graph Convolutional Network (GCN) model [32] with 2 layers of 64 units each for 100 epochs. We apply dropout to hidden units with a factor of 0.3 to stabilize the training process.

We present the results on Figure 6. We can observe that on most datasets tree-based sparsification methods outperform other baselines. Compared to graph clustering and embedding, graph neural networks are more robust to disconnected components—in fact, GNNs are less sensitive to structure of graphs overall, since these models have features to rely on. Therefore, differences between methods are less pronounced for this graph learning approach. However, we can still reap the benefits of tree-based sparsification: kTree is consistently a top performer.

We obtain sizeable benefits in sparsifying graphs for GNNs. On all datasets, graph neural networks obtain performance comparable or better than the full graph at average degree equal to $\bar{d} = 5$, when this level of sparsification was available. This point is obtained at slightly lower sparsity levels than for graph clustering and embedding, which can be explained by the fact that GNNs smooth the information via graph structure, and that process works best with more connections on average.

5.7 General Observations and Trends

Overall, our extensive experimental study suggests that finding very sparse GLT winners is possible. Our algorithms are able to offer significant improvements compared to baselines in terms of six graph structure quality metrics introduced in Section 3.

On three distinct graph learning problems, we have showed that it is possible to obtain comparable *or better* performance than the original graph structure with average node degree in the range 2–5. Importantly, we show considerable performance improvements on graphs constructed from data.

6 Conclusion

This work postulates the GLT hypothesis that states that extremely sparse backbones allow various graph learning algorithms to attain comparable performance as on the full graph. We suggest two efficient algorithms to uncover such "winning tickets". Our experimental results illustrate our methods' effectiveness, matching the performance of different graph learning algorithms in very sparse graphs ($\approx$ average degree of 5). Extensions to bipartite graphs are of immediate interest since bipartite interaction graphs suffer from various problems with high-degree "celebrity" nodes.

References

[1] Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 1993. Cited on page 2.

[2] Reid Andersen, Fan Chung, and Kevin Lang. Using pagerank to locally partition a graph. *Internet Mathematics*, 2007. Cited on page 3.

[3] Adrián Arnaiz-Rodríguez, Ahmed Begga, Francisco Escolano, and Nuria Oliver. Diffwire: Inductive graph rewiring via the lovász bound. In *LoG*, 2022. Cited on page 3.

[4] Pradeep Kr Banerjee, Kedar Karhadkar, Yu Guang Wang, Uri Alon, and Guido Montúfar. Oversquashing in GNNs through the lens of information contraction and graph expansion. In *58th Annual Allerton Conference on Communication, Control, and Computing*. IEEE, 2022. Cited on page 3.

[5] Joshua D Batson, Daniel A Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. In *STOC*, 2009. Cited on page 2.

[6] András A Benczúr and David R Karger. Approximating st minimum cuts in $\tilde{O}(n^2)$ time. In *STOC*, 1996. Cited on page 2.

[7] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008. Cited on pages 5 and 7.

[8] Chen Cai and Yusu Wang. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*, 2020. Cited on page 3.

[9] CJ Carey, Jonathan Halcrow, Rajesh Jayaram, Vahab Mirrokni, Warren Schudy, and Peilin Zhong. Stars: Tera-scale graph building for clustering and learning. In *NeurIPS*, 2022. Cited on page 2.

[10] Hau Chan and Leman Akoglu. Optimizing network robustness by edge rewiring: a general framework. *DMKD*, 2016. Cited on pages 2, 6, and 7.

[11] Sudhanshu Chanpuriya, Cameron Musco, Konstantinos Sotiropoulos, and Charalampos Tsourakakis. Node embeddings and exact low-rank representations of complex networks. *NeurIPS*, 2020. Cited on page 4.

[12] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: fast learning with graph convolutional networks via importance sampling. In *ICLR*, 2018. Cited on page 3.

[13] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *ICML*, 2021. Cited on page 3.

[14] Fan RK Chung. *Spectral graph theory*. AMS, 1997. Cited on page 3.

[15] L da F Costa, Francisco A Rodrigues, Gonzalo Travieso, and Paulino Ribeiro Villas Boas. Characterization of complex networks: A survey of measurements. *Advances in physics*, 2007. Cited on page 4.

[16] Andreea Deac, Marc Lackenby, and Petar Veličković. Expander graph propagation. *arXiv preprint arXiv:2210.02997*, 2022. Cited on page 4.

[17] Karel Devriendt and Renaud Lambiotte. Discrete curvature on graphs from the effective resistance. *Journal of Physics: Complexity*, 2022. Cited on page 4.

[18] Laxman Dhulipala, David Eisenstat, Jakub Łącki, Vahab Mirrokni, and Jessica Shi. Hierarchical agglomerative graph clustering in nearly-linear time. In *ICML*, 2021. Cited on page 1.

[19] Navid Dianati. Unwinding the hairball graph: Pruning algorithms for weighted complex networks. *Physical Review E*, 2016. Cited on pages 6 and 7.

[20] Robin Forman. Bochner's method for cell complexes and combinatorial ricci curvature. *Discrete and Computational Geometry*, 29(3):323–374, 2003. Cited on page 4.

[21] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019. Cited on page 1.

[22] Wai Shing Fung, Ramesh Hariharan, Nicholas JA Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *STOC*, 2011. Cited on page 2.

[23] Wai Shing Fung and Nicholas JA Harvey. Graph sparsification by edge-connectivity and random spanning trees. *arXiv preprint arXiv:1005.0265*, 2010. Cited on page 5.

[24] Johannes Gasteiger, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. In *NeurIPS*, 2019. Cited on page 3.

[25] Arpita Ghosh and Stephen Boyd. Growing well-connected graphs. In *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE, 2006. Cited on page 3.

[26] Navin Goyal, Luis Rademacher, and Santosh Vempala. Expanders via random spanning trees. In *SODA*. SIAM, 2009. Cited on page 5.

[27] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017. Cited on page 3.

[28] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020. Cited on page 13.

[29] Jürgen Jost and Shiping Liu. Ollivier's ricci curvature, local clustering and curvature-dimension inequalities on graphs. *Discrete & Computational Geometry*, 2014. Cited on page 4.

[30] David R Karger. Using randomized sparsification to approximate minimum cuts. In *SODA*, 1994. Cited on page 2.

[31] Kedar Karhadkar, Pradeep Kr Banerjee, and Guido Montúfar. FoSR: First-order spectral rewiring for addressing oversquashing in gnns. In *ICLR*, 2023. Cited on pages 3, 6, and 7.

[32] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. Cited on pages 5 and 9.

[33] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. Cited on page 13.

[34] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 1956. Cited on page 2.

[35] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. Cited on page 13.

[36] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 2003. Cited on page 7.

[37] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 2006. Cited on page 6.

[38] Andrew Ng, Michael Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *NIPS*, 2001. Cited on page 3.

[39] Krzysztof Nowicki and Tom A B Snijders. Estimation and prediction for stochastic blockstructures. *Journal of the American statistical association*, 2001. Cited on page 6.

[40] Yann Ollivier. Ricci curvature of markov chains on metric spaces. *Journal of Functional Analysis*, 256(3):810–864, 2009. Cited on page 4.

[41] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *ICLR*, 2020. Cited on page 3.

[42] John Palowitch, Anton Tsitsulin, Brandon Mayer, and Bryan Perozzi. Graphworld: Fake graphs bring real insights for gnns. In *KDD*, 2022. Cited on page 1.

[43] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *JMLR*, 2011. Cited on page 8.

[44] David Peleg and Alejandro A Schäffer. Graph spanners. *Journal of graph theory*, 1989. Cited on page 2.

[45] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014. Cited on pages 3, 5, and 8.

[46] Areejit Samal, RP Sreejith, Jiao Gu, Shiping Liu, Emil Saucan, and Jürgen Jost. Comparative analysis of two discretizations of ricci curvature for complex networks. *Scientific reports*, 8(1):1–16, 2018. Cited on page 4.

[47] Aaron Schild. An almost-linear time algorithm for uniform random spanning tree generation. In *STOC*, 2018. Cited on page 5.

[48] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008. Cited on page 13.

[49] C Seshadhri, Aneesh Sharma, Andrew Stolman, and Ashish Goel. The impossibility of low-rank representations for triangle-rich complex networks. *Proceedings of the National Academy of Sciences*, 117(11):5631–5637, 2020. Cited on pages 1 and 4.

[50] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018. Cited on pages 6 and 13.

[51] Daniel A Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *STOC*, 2008. Cited on page 2.

[52] Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 2011. Cited on page 2.

[53] Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M Bronstein. Understanding over-squashing and bottlenecks on graphs via curvature. In *International Conference on Learning Representations*, 2022. Cited on pages 1, 3, and 4.

[54] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alexander Bronstein, and Emmanuel Müller. Spectral graph complexity. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 308–309, 2019. Cited on pages 1 and 3.

[55] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *WWW*, 2018. Cited on page 3.

[56] Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of tr(f(a)) via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 2017. Cited on pages 4 and 6.

[57] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 1998. Cited on pages 4 and 7.

[58] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. Cited on page 13.

[59] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 1977. Cited on page 1.

[60] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, 2003. Cited on page 3.

A Appendix.

A.1 Dataset description

Here we present a brief description of real-world datasets:

- Cora, Citeseer, and Pubmed [48] are citation networks; nodes represent papers connected by citation edges; features are bag-of-word abstracts, and labels represent paper topics. We use a re-processed version of Cora from [50] due to errors in the processing of the original dataset.
- Amazon {PC, Photo} [50] are two subsets of the Amazon co-purchase graph for the computers and photo sections of the website, where nodes represent goods with edges between ones frequently purchased together; node features are bag-of-word reviews, and class labels are product category.
- OGB-ArXiv [28] is a paper co-citation dataset based on arXiv papers indexed by the Microsoft Academic graph. Nodes are papers; edges are citations, and class labels indicate the main category of the paper.
- CIFAR, MNIST, and FashionMNIST [33, 35, 58] are ε-nearest neighbor graphs with ε such that the average node degree is 100.

Table 1: Dataset statistics. We report total number of nodes $|V|$, average node degree $\bar{d}$, number of features $|X|$ and labels $|Y|$.

dataset	$\lvert V\rvert$	$\bar{d}$	$\lvert X\rvert$	$\lvert Y\rvert$
Cora	19793	3.20	1433	7
Citeseer	3327	1.37	3703	6
PubMed	19717	2.25	500	3
Amazon PC	13752	17.88	767	10
Amazon Photo	7650	15.57	745	8
MSA-Physics	34493	7.19	8415	5
OGB-arXiv	169343	6.84	128	40
CIFAR-10	50000	99	3072	10
FashionMNIST	60000	99	784	10
MNIST	60000	99	784	10

A.2 Metrics on Real-World Datasets

Here we present graph metrics computed on real-world graphs present in our experimental study.

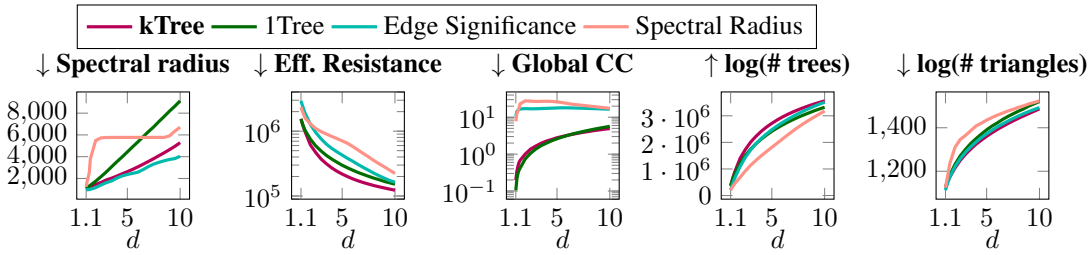

Figure 7: Graph statistics measured on the AmazonPC graph.

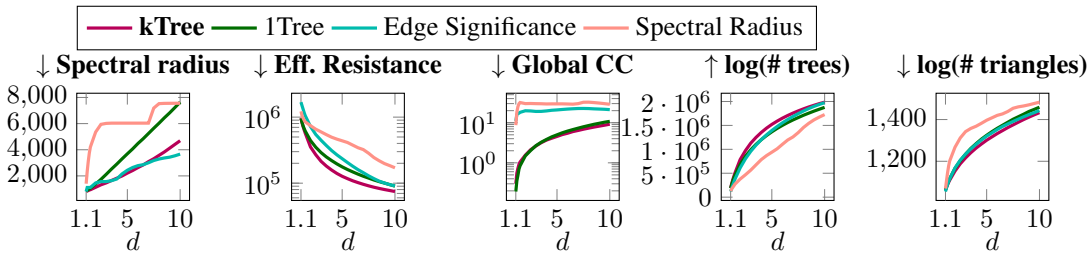

Figure 8: Graph statistics measured on the AmazonPhoto graph.

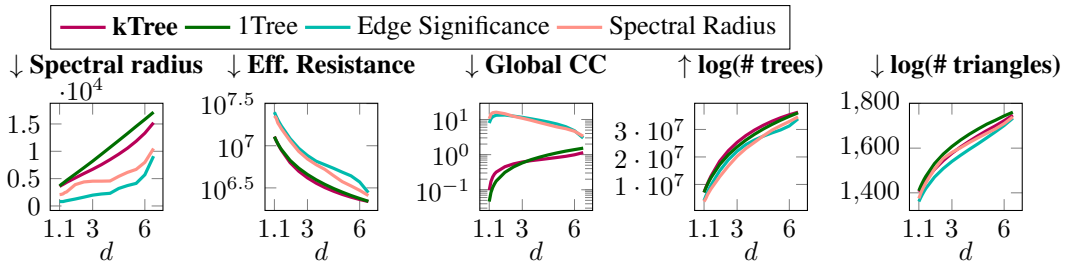

Figure 9: Graph statistics measured on the OGB-ArXiv graph.

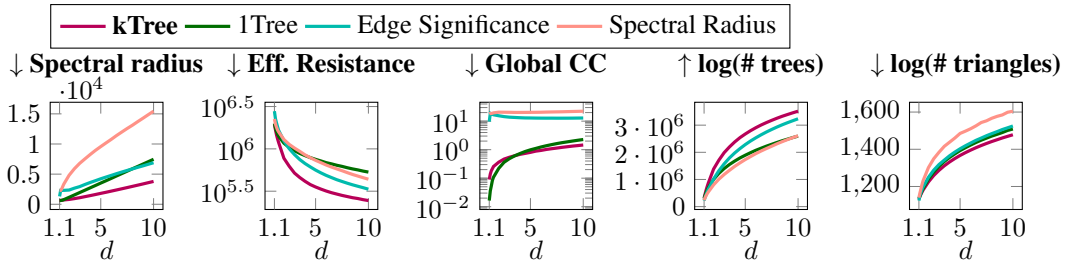

Figure 10: Graph statistics measured on the CIFAR10 graph.

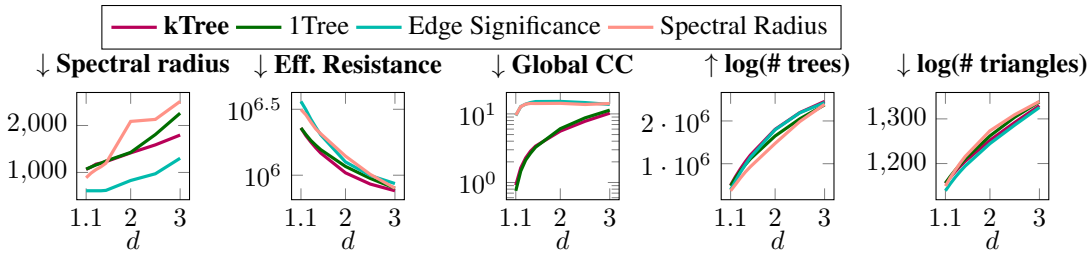

Figure 11: Graph statistics measured on the Cora graph.

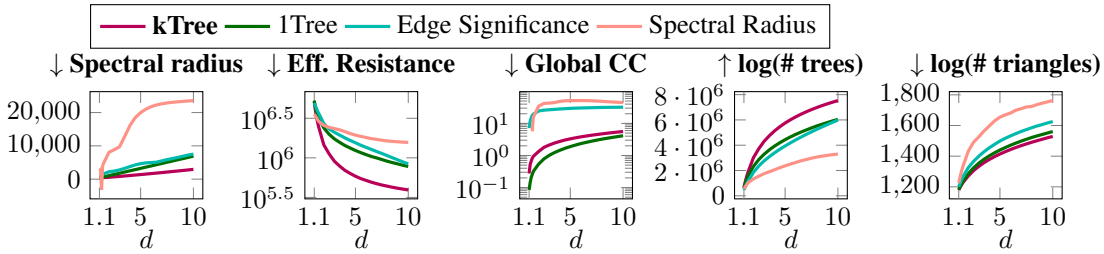

Figure 12: Graph statistics measured on the FashionMNIST graph.

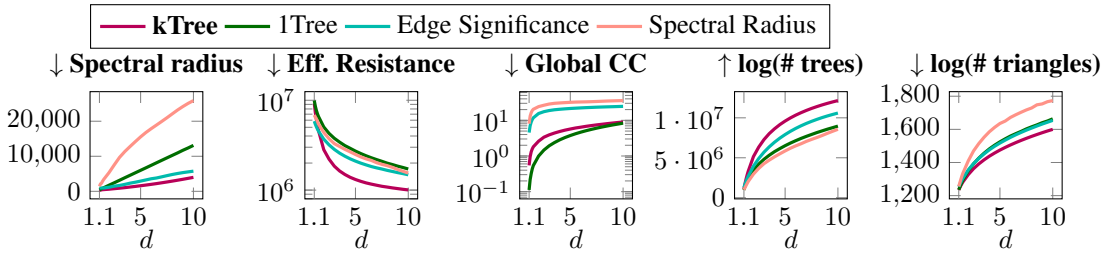

Figure 13: Graph statistics measured on the MNIST graph.

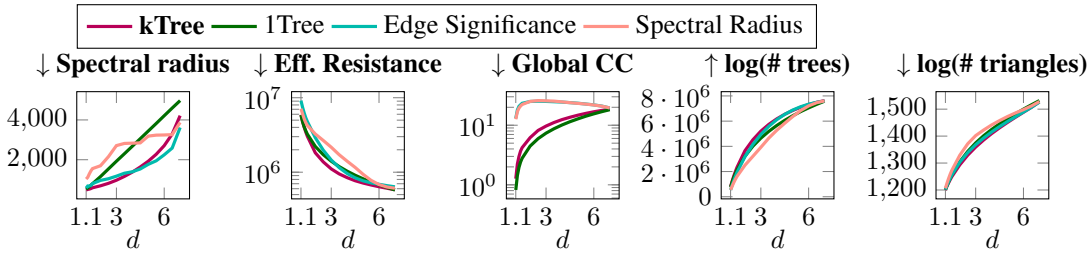

Figure 14: Graph statistics measured on the MSA-Physics graph.

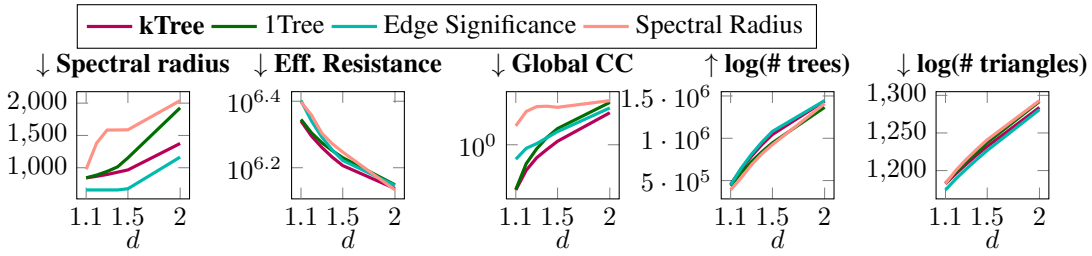

Figure 15: Graph statistics measured on the Pubmed graph.