TAMARA: a task-aware multilayer graph simplification framework

Cheick Tidiane Ba[1][0000−0002−4035−7464], Roberto Interdonato[2][0000−0002−0536−6277], Dino Ienco[3][0000−0002−8736−3132], and Sabrina Gaito[1][0000−0003−3779−2809]

[1] University of Milan, Milan, Italy {cheick.ba,sabrina.gaito}@unimi.it
[2] CIRAD, UMR TETIS, Montpellier, France roberto.interdonato@cirad.fr
[3] INRAE, UMR TETIS, Montpellier, France dino.ienco@inrae.fr

Abstract. In many scientific fields, accurately representing the heterogeneous relationships that may characterize a complex system, has contributed to the success of multilayer graph models. However, modeling such complex information can be challenging in real-world cases. On one hand, including all relationships may lead to noisy or computationally intensive graphs. On the other hand, selecting a limited number of relationships can result in boundary specification problems. To address these challenges thus improving and optimizing analysis and visualization tasks, only task-agnostic preprocessing techniques that depend on unsupervised heuristics are presently used. However, from task to task, there can be substantial differences in the graph features that should be preserved. A task-agnostic strategy risks failing because it cannot adapt. Deep learning methods for single-layer graphs have recently been introduced, demonstrating how effectively the simplification task can be included in the overall training process. In this work, we propose the Task Aware MultilAyer gRaph simplificAtion (TAMARA) framework, a GNN-based approach designed to simplify multilayer graphs based on the downstream task. TAMARA generates node embeddings for a specific task by training jointly two main components: i) an edge simplification module and ii) a (multilayer) graph neural network. We tested TAMARA on different real-world multilayer graphs for node classification tasks. Experimental results show the effectiveness of the proposed approach: TAMARA reduces the dimension of the input graph while keeping and even improving the performance of node classification tasks in different domains and across graphs characterized by different structures. Moreover, deep learning-based simplification allows TAMARA to preserve and enhance graph properties important for the task. To our knowledge, TAMARA represents the first simplification framework for machine learning on multilayer graphs.

Keywords: graph neural network · graph simplification · Multilayer graph

1 Introduction

The graph analysis and mining research field has raised in popularity in the last two decades, thanks to the ability of graphs to model a wide range of real-life phenomena from physical [1] to biological [22] and social systems [18], from scientific [19] to financial data [31,4], transportation routes [6], and many others [5]. In this regard, the multilayer graph model [13] is widely used as a powerful tool to represent the organization and relationships of complex systems covering many different domains. Multilayer graphs are designed to provide a more realistic representation of the different and heterogeneous relationships that may characterize an entity in the graph-structured system, using the rich data available from complex systems [10].

However, collecting a wide set of different relationships among a large set of entities can easily result in a significant amount of noise (e.g., incomplete, imprecise or redundant information) caused by the choice regarding which entities and relations should be included in the data. Single-layer graphs already have this issue, known as the boundary specification problem [7], which is exacerbated in multilayer ones. While for the simplification of single-layer graphs, several machine learning techniques have already been proposed in the literature and have proved to be effective [32,25], for multilayer graphs there are only unsupervised heuristics of preprocessing [10], while cutting-edge techniques such as graph neural networks have not yet been exploited. Furthermore, work on multilayer graph neural networks [21,28] demonstrated how crucial it is to conceive approaches specially tailored for these complex structures, i.e., to produce embeddings that convey the rich information present in the input graph. As a matter of fact, the direct application of single-layer approaches to multilayer graphs is not trivial: while a single-layer approach could be applied on each layer separately, the important interplay among the various layers would be lost. The same holds for the simplification task at the heart of this work: a framework able to thoroughly leverage the multilayer structure is of paramount importance to obtain a simplified object properly optimized for a given downstream task.

In this work, we propose the Task Aware MultilAyer gRaph simplificAtion (TAMARA) framework, a GNN-based framework designed to simplify multilayer graphs based on the downstream task. TAMARA generates node embeddings for a specific task by training end-to-end two main components: i) an edge simplification module and ii) a (multilayer) graph neural network. We tested TAMARA under node classification on real-world multilayer graphs from different domains. Experimental results show the effectiveness of the proposed approach: TAMARA dramatically reduces the dimension of the input graph not only maintaining but also improving the performance. In addition, TAMARA provides different approaches allowing us to provide insights on the most effective simplification strategy depending on the domain of the downstream task. In fact, with TAMARA, we enable simplification approaches that leverage single-layer simplification techniques on multilayer graphs but we also extend existing methods to work directly on multilayer graphs. Thus, TAMARA can select the appropriate simplification approach depending on the task. Moreover, we observe that deep learning driven

simplification with TAMARA can influence and enhance important graph properties, such as label assortativity: as the selection of task-irrelevant edges is refined during the training, TAMARA is guided in the selection of the most important properties to preserve or enhance. To our knowledge, TAMARA represents the first GNN-based simplification framework for multilayer graphs.

Due to the wide range of data that can be modelled as a multilayer graph, the proposed framework can have a large application room covering different fields like biology, physics, and health/medical analysis, where increased robustness is needed to address noise from data acquisition. Furthermore, data quality, computational performances and information visualization are also crucial aspects of any process dealing with massive amounts of graph-structured data, such as social media mining, communication, biological, transportation and financial systems.

2 Background

In this section, we provide background knowledge regarding the formal definition of the multilayer graph model adopted in this paper, the use of graph neural networks for the analysis of multilayer graphs, and graph simplification approaches based on deep learning.

Multilayer graph model. Since in this work we will use ML-GCN (Multilayer Graph Convolutional Neural Network) to instantiate TAMARA, for ease of reference we adopt the same definition of multilayer graph as in the work where it was originally proposed [28]:

Definition 1 (Multilayer graph). *Given a set $\mathcal{V}$ of entities, and a set of layers $\mathcal{L} = \{L_1, ...L_l\}$ with $\mid \mathcal{L} \mid = L >= 2$, a multilayer graph is $\mathcal{G}_\mathcal{L} = (\mathcal{V}_\mathcal{L}, \mathcal{E}_\mathcal{L}, \mathcal{V}, \mathcal{L})$, where $\mathcal{V}_\mathcal{L} \subseteq \mathcal{V} \times \mathcal{L}$ is the set of entity-layer pairings or nodes (i.e., to denote which entities are present in which layers), and $\mathcal{E}_\mathcal{L} = \mathcal{V}_\mathcal{L} \times \mathcal{V}_\mathcal{L}$ is the set of directed edges between nodes within and across layers.*

Definition 2 (Within-layer edges/links). *Links connecting nodes in the same graph layer $((i,l),(j.m)) \mid (i,l),(j.m) \in \mathcal{E}_\mathcal{L}, l = m$*

Within-layer edges can be described by a set of adjacency matrices $A = \{A_1, ..., A_\ell\}$. These adjacency matrices describe layer-by-layer connections. But, in multilayer, we also have cross-layer connections.

Definition 3 (Cross-layer edges/links). *Links connecting nodes in different layers $((i,l),(j.m)) \mid (i,l),(j.m) \in \mathcal{E}_\mathcal{L}, l \neq m$*

Definition 4 (Pillar edges/links). *Cross-layer links connecting nodes with the same index but in different layers $((i,l),(j.m)) \mid (i,l),(j.m) \in \mathcal{E}_\mathcal{L}, i = j, l \neq m$*

Hence, we can define a Supra adjacency matrix

Definition 5 (Supra adjacency matrix). *The supra adjacency matrix* A^{sup} *is:*

$$A^{sup} = \begin{cases} A_l \text{ if diagonal block} \\ A_{l,m} \text{ otherwise (i.e., off the diagonal block).} \end{cases} \quad (1)$$

where $A_{l,m}$ *is an inter-layer adjacency matrix built upon the inter-layer connections between layer* l *and layer* m *(i.e.,* 1 *if there exists an edge between* (i,l) *and* (u,m) *with* $l = m$*, and* 0 *otherwise).*

Graph neural networks. In the field of deep learning for graph-structured data, graph neural networks (GNNs) have emerged as the state-of-the-art approach in many different tasks, such as node classification [8], link prediction [29], community detection [27] and graph classification[30]. GNNs redefine basic deep learning operations, such as convolution, for graph-structured data. In the Graph Convolutional Network (GCN) model proposed by [12], the operation of convolution on graphs is performed through an aggregation of the values of each node's features along with its neighbors' features. The graph attention network model (GAT) [23] learns weights between each pair of connected nodes; a self-attention mechanism is used to discover the most representative parts of the input. Starting from these, we have seen the proposal of many architectures, to cover different tasks and types of graph data such as signed graphs, temporal graphs, and more recently multilayer graphs.

Graph neural networks for multilayer graphs. Deep learning tasks are more challenging on these graphs because of the presence of intra-layer and inter-layer relations, different layer characteristics, as well as node features. There have been some attempts to design methods and frameworks for deep learning for multilayer graphs. State-of-the-art results have been obtained by the framework presented in [28]. The framework reformulates the propagation rule of the GNN component (i.e. GCN or GAT) to aggregate topological neighborhood information from different layers. While in GCN, aggregation involves a node's features and its neighbors' features, in the ML-GCN the aggregation is performed with both its neighbors in that layer (dubbed within-layer neighborhood) and on its neighbors located in other layers where the entity occurs (referred to as outside-layer neighborhood). More formally:

$$h_{(i,l)}^{(k+1)} = \sigma\left(\sum_{(j,m)\in\Gamma(i,l)\cup\Psi(i,l)} \frac{1}{\sqrt{\widetilde{D}_{ii}\widetilde{D}_{jj}}} h_{(j,m)}^{(k)} W^{(k+1)}\right) \quad (2)$$

Deep learning for graph simplification. Graph simplification consists in removing uninformative or redundant edges while keeping almost all information of the input graph [20]. While there are many works on simplification [15], only a few are focused on simplification for deep learning on graphs. Dropedge [20] simplifies the graph for a GNN model (e.g. GCN, GAT) by randomly removing a fraction of the edges from the input graph during the training phase. The evaluation of Dropedge shows that even a random removal can lead to an improvement

of performance across different tasks, such as node classification and link prediction. In NeuralSparse [32], the simplification process is done through the deep neural network: during the training phase, the deep neural network learns a simplification strategy that favors downstream tasks. In the testing phase, that neural network is used to select the edges to remove from the input graph, based on the learned strategy. Other works rely on similar principles. In AdaptiveGCN [14] simplification process is led by a deep neural network like in NeuralSparse, but a simplification step is performed before each graph convolution step. In PTDnet [16] additional constraints on the simplification process are introduced, encouraging the removal of more edges or prioritizing the simplification of edges connecting different node clusters. Other works such as [24] and [25] have designed frameworks for simplification with reinforcement learning. While there are several works on single-layer graph simplification, there is a lack of work relying on deep learning for the simplification of multilayer graphs.

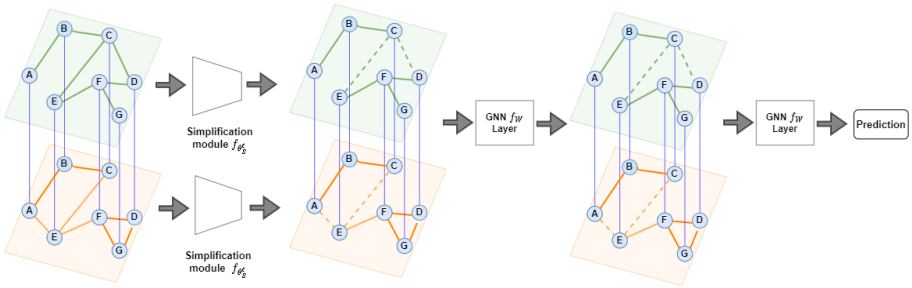
3 The TAMARA framework

Problem definition. In this work, we want to provide a framework for the simplification of multilayer graphs and evaluate the impact of a simplification approach on a machine learning task. We want to evaluate the impact of graph simplification approaches on a typical machine learning task, i.e., node classification. The graph simplification problem on single-layer graphs can be defined as follows: given a graph $G(V, E, X_E, X_V)$, where V is a set of n nodes, $E \subset V \times V$ is the set of edges; X_V is a set of node attributes, X_E is a set of edge attributes. simplification tries to obtain a subgraph of G, that would be $G' = G(V', E', X_E, X_V)$, where $V' \subset V \vee E' \subset E$. Whereas on a multilayer graph, simplification can be defined as the problem of obtaining a graph $f_{\theta_S}(\mathcal{G}_\mathcal{L}) = \mathcal{G}_\mathcal{L}{}' = (\mathcal{V}_\mathcal{L}{}', \mathcal{E}_\mathcal{L}{}', \mathcal{V}', \mathcal{L}')$ such that the following disjunction of conditions holds: $\mid \mathcal{V} \mid < \mid \mathcal{V}' \mid \vee \mid \mathcal{L} \mid < \mid \mathcal{L}' \mid \vee \mid \mathcal{V}_\mathcal{L} \mid < \mid \mathcal{V}_\mathcal{L}{}' \mid \vee \mid \mathcal{E}_\mathcal{L} \mid < \mid \mathcal{E}_\mathcal{L}{}' \mid$.

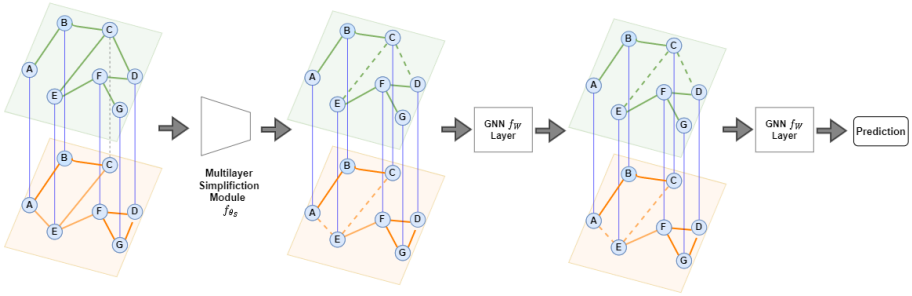
Simplification approaches. In order to perform graph simplification on a multilayer graph, we propose two approaches: i) Layer by layer graph simplification and ii) Multilayer graph simplification. We now present the two concepts behind them.

Layer by layer graph simplification. To perform graph simplification on a multilayer graph by exploiting methods for single-layer graphs, we can use a layer-by-layer approach. In the layer-by-layer simplification, methods are applied to each layer before recomposing the supra-adjacency matrix: cross-layer links are not involved. We can define a layer graph as $G[\ell]$ where every edge connects nodes in the same layer ℓ. Therefore, at each layer ℓ a simplification neural network $f_{\theta_S^\ell}$ detects noisy links over the layer-graph $G[\ell]$, generating a new version of the graph that we can define as $G[\ell]'$. The simplified graphs are used to update A'^{sup}, which will be used to train the graph neural network.

It's important to note that simplification can be applied at a different *stage*s of the process: we can simplify *once* or before *each* graph convolutional layer.



(a) *Layer by layer graph simplification with multilayer GNN.* A simplification module (simplification neural network $f_{\theta_S^\ell}$) detects the links to remove at each layer ℓ of the input multilayer graph, while a GNN (multilayer graph neural network f_W) generates embeddings for a downstream task.



(b) *Multilayer graph simplification with multilayer GNN.* A multilayer simplification module (simplification neural network f_θ) detects the links to remove by taking into account the whole input multilayer graph, while a GNN (multilayer graph neural network f_W) is used to generate node embeddings for a downstream task.

Fig. 1: Overview of the proposed approaches for multilayer graph simplification: (a) *layer-by-layer* and (b) *multilayer*. Note that the difference between the two approaches lies in the simplification process, while the use of the GNN is the same.

In the first case, a simplification module detects noisy elements while a graph neural network model is used to generate node embeddings for a downstream task. Here, simplification occurs only *once*, so that the graph is the same at each GNN layer. In the other case, at each GNN layer, a simplification module detects noisy elements while a graph neural network model generates node embeddings for a downstream task. The simplification is performed multiple times so that before *each* GNN layer, we are working on different versions of the graph.

TAMARA allows training with both simplification stages. The training phase for layer-by-layer graph simplification is summarized in algorithm 1.

Multilayer graph simplification. To define a simplification methodology conceived explicitly for a multilayer graph, able to properly take into account the

complex structure of such models, we propose to use a simplification neural network f_θ that detects noisy edges and a graph neural network f_W to generate node embeddings for a downstream task (cf. Fig. 1b). The key difference with respect to the single-layer counterpart is that the simplification module is unique, and acts directly on the supra-adjacency matrix A^{sup} to generate the simplified A'^{sup}. Working directly on the supra-adjacency matrix also has an additional advantage: the simplification module can remove noisy or redundant cross-layer links as well. Even in the multilayer simplification case, simplification can be applied at different *stage*s: we can simplify *once* (i.e., the graph is the same at each GNN layer) or before *each* graph convolutional layer (i.e., the simplification is performed multiple times, so that each GNN layer works on a different version of the graph). The training phase for multilayer graph simplification is summarized in algorithm 2.

Algorithm 1 Training algorithm for layer-by-layer simplification with ML GNN

Input: training graph $G(V, E, X_E, X_V)$, $\mathcal{L}$ multilayer graph layers, simplification neural network f_{θ_S}, simplification stage *stage*, number of GNN hidden layers K
Output: Embeddings for downstream task
if *stage* = "once" **then** ▷ Simplify graph just once
  for layer $\ell \in 1...\mathcal{L}$ **do**
    $A'_\ell \leftarrow f_{\theta^\ell_S}(A_\ell)$ simplification function applied on $G[\ell]$
  end for
  $A'^{sup} \leftarrow$ Combine A'_ℓ in supra adjacency matrix
end if
for $k = 1...K$ **do**
  if *stage* = "each" **then** ▷ Different graph every time
    for layer $\ell \in 1...\mathcal{L}$ **do**
      $A'_\ell \leftarrow f_{\theta^\ell_S}(A_\ell)$ simplification function applied on $G[\ell]$
    end for
    $A'^{sup} \leftarrow$ Combine A'_ℓ in supra adjacency matrix
  end if
  $H^k \leftarrow f_W^{(k-1)}(H^{(k-1)}, A'^{sup})$ ▷ hidden representations update
end for
Backpropagation to update θ_W, θ^ℓ_S

4 Experimental evaluation

Data. For the experimental evaluation, we selected datasets from different domains showing different structural characteristics, summarized in Table 1. All of them correspond to multilayer graphs with associated real-world node features, a characteristic that is crucial for the proposed task-aware simplification process. The ***um-econ*** and ***um-socioeco*** [2] multilayer graphs describe user interactions in a decentralized social media platform (Steemit) [3]. In these graphs nodes are users, and layers are interactions of different types. User features are graph-based metrics. User labels describe their migration to another social media platform, called Hive (4 cases: inactive, stay, leave, active on both). In *um-econ* is a subgraph composed of 2 layers of economic interactions, while *um-socioeco* considers interaction on 4 layers, 2 social and 2 economic. ***IMDb-mlh*** [17] is a multilayer

Algorithm 2 Training algorithm for multilayer simplification with ML GNN

Input: training graph $G(V, E, X_E, X_V)$, $\mathcal{L}$ multilayer graph layers, simplification neural network f_{θ_S}, simplification stage *stage*, number of GNN hidden layers K
Output: Embeddings for downstream task
if *stage*= "once" **then** ▷ Simplify graph just once
    $A'^{sup} \leftarrow f_{\theta_S}(A^{sup})$ ▷ Simplify
end if
for $k = 1...K$ **do**
    if *stage* = "each" **then** ▷ Different graph every time
        $A'^{sup} \leftarrow f_{\theta_S}(A^{sup})$ ▷ Simplify
    end if
    $H^k \leftarrow f_W^{(k-1)}(H^{(k-1)}, A'^{sup})$ ▷ hidden representations update
end for
Backpropagation to update θ_W, θ_S

graph constructed from the IMDb movie database, where nodes are movies, and two movies are connected if they share either an actor or a director. Movie features encode text from the plots, while the labels describe the movie type (action, comedy, drama). Finally ***Koumbia 2*** and ***Koumbia 5*** [9,28] are multilayer graphs extracted from a time series of Sentinel-2[4] optical satellite images, covering the agricultural landscape of Koumbia in Burkina Faso. Nodes represent segments of the satellite image, and labels correspond to either crop (cultivated areas) or no-crop (uncultivated areas, such as forests) segments. Layers correspond to functional classes (e.g., temporal radiometric profiles). The network includes inter-layer edges and real-world attributes, corresponding to time series of radiometric statistics for each segment. The graphs are generated with the geo2net framework[5], which allows the production of multilayer graphs from satellite images with an arbitrary number of layers: in this work, we consider 2 and 5 layers.

Table 1: Summary of structural characteristics of the graph datasets: type of the graph, number of layers (L), number of nodes ($|V|$), number of edges ($|E|$), density (mean/SD) over the layers (d), and number of classes (C)

dataset	L	$\|V\|$	$\|E\|$	d	C
imdb-mlh	2	5614	23208	0.0007 ± 0.0000	3
um-econ	2	15414	224855	0.0018 ± 0.0012	4
um-socioeco	4	18212	1199863	0.0138 ± 0.0118	4
Koumbia 2	2	4492	18783	0.0010 ± 0.0001	2
Koumbia 5	5	11230	91938	0.0010 ± 0.0002	2

Experimental setting. In this work, we focus on node classification tasks, i.e., we learn the embeddings required to predict the label associated to each node in the graph. As GNN for TAMARA we select the GCN, but note that other GNNs could be employed. As a baseline, we consider a multilayer GNN without

[4] `https://sentinel.esa.int/web/sentinel/missions/sentinel-2`
[5] `https://gitlab.irstea.fr/raffaele.gaetano/geo2net`

simplification (***GNN***). As previously discussed, TAMARA is flexible and can be equipped with different simplification strategies as well. In this work, we selected i) DropEdge [20] (TAMARA(DE)), a single-layer graph simplification method that randomly removes edges with probability p, and ii) NeuralSparse [32] (TAMARA(NS)), which is able to leverage node features to select a subset of edges to keep (a subgraph-based selection process is performed where for each node only k of its neighbors are kept and their connecting edges). Note that both approaches were originally designed for single-layer simplification, hence for this work, we implemented extended versions in order to perform multilayer (*multi*) and layer-by-layer (*l-b-l*) simplification (cf. Section 3). Moreover, each implementation can be applied at different *stages*: we can simplify *once* or before *each* graph convolution layer (cf. Section 3). For TAMARA(DE), we test different drop rate probabilities $p = \{0.1, 0.3, 0.5, 0.7\}$, while for TAMARA(NS), we test different $k = \{5, 10.15\}$, with τ varying during training as in [32]. We perform all the experiments with a transductive learning setting like in [28]. In a transductive setting, all node attributes and topological information can be used for training, while only a subset of labels is visible to the GNN model. All models were trained using the Adam optimization algorithm [11] with full batch training [12], L2 weight regularization set to 0.0005. For each graph and method, the average accuracy was computed over $N = 3$ independent runs, where each run corresponded to a different train-validation-test split, with 25% of training entities as previously done in [28] and the rest split in validation (25%) and test entities (50%). The combination of hyperparameters with the best average validation metric is selected, and we report the final test metric. Since we are working on a huge number of possible combinations, we rely on early stopping, training for 250 epochs with 10 epochs of patience (reloading the best model). As an evaluation metric, we select AUC (Area under the ROC Curve) evaluated like in [32], because it is well suited for datasets showing unbalanced label distribution, such as *Imdb*, *um-econ* and *um-socioeco*.

5 Results

Framework evaluation. Table 2 reports the average AUC scores on the test set. We can observe how TAMARA generally improves upon the GNN baseline, and always corresponds to the best performances. Note that TAMARA(NS) almost consistently outperforms TAMARA(DE), demonstrating the importance of exploiting node features for the simplification task. The only exception is represented by *imdb-mlh*, where features information improves the performance, but the TAMARA(DE) variant obtains even better performance. Additional insights can be obtained by comparing the multilayer (*multi*) vs layer-by-layer (*l-b-l*) and the *once* vs *each* approaches. Regarding TAMARA(DE), we note that *multi* tends to be more effective on 2-layer graphs (i.e., *um-econ*, *um-socioeco* and *Koumbia-2*) while *l-b-l* seems to be more effective in presence of a greater number of layers. Note also that, with the (DE) variant, simplifying *once* tends to be the winning choice. This is consistent with the stochastic nature of this approach,

i.e., repeating a random process at each layer may negatively impact the result. As concerns TAMARA(NS), *l-b-l* tends to be the best choice in most cases: it may be because the NeuralSparse simplification is based around a single-layer notion of a node's neighborhood. Devising an advanced strategy to properly take into account the multilayer neighborhood is left as future work. In terms of when to simplify (stage), for the task-aware (NS) variant, we can see that simplifying *once* brings better results for datasets showing an unbalanced distribution of the labels (i.e., *um-econ*, *um-socioeco* and *imdb-mlh*), while simplifying before *each* convolution layer seems the best approach for the more balanced *Koumbia* graphs.

Overall, TAMARA leads to significant performance improvements, while the variety of proposed approaches allows TAMARA to find the most suitable simplification approach for tasks of different domains.

Table 2: AUC (mean and standard deviation over 3 random seeds [26]) obtained by the baseline and TAMARA.

model	simp	data stage	um-econ	um-socioeco	imdb-mlh	Koumbia 2	Koumbia 5
GNN	-	-	0.7420 ± 0.0022	0.6939 ± 0.0234	0.8035 ± 0.0218	0.9056 ± 0.0049	0.9237 ± 0.0033
TAMARA (DE)	multi	once	0.7451 ± 0.0128	0.6936 ± 0.0279	**0.8135 ± 0.0351**	0.9068 ± 0.0007	0.9228 ± 0.0041
		each	0.7487 ± 0.0150	0.6905 ± 0.0233	0.8122 ± 0.0324	0.9042 ± 0.0075	0.9246 ± 0.0069
	l-b-l	once	0.7407 ± 0.0083	0.6939 ± 0.0234	0.8005 ± 0.0253	0.9059 ± 0.0059	0.9252 ± 0.0063
		each	0.7418 ± 0.0102	0.6988 ± 0.0130	0.8079 ± 0.0280	0.9022 ± 0.0045	0.9238 ± 0.0051
TAMARA (NS)	multi	once	**0.7522 ± 0.0084**	0.6924 ± 0.0208	0.8011 ± 0.0299	0.9023 ± 0.0042	0.9223 ± 0.0138
		each	0.7458 ± 0.0107	0.6817 ± 0.0347	0.7987 ± 0.0257	0.9080 ± 0.0023	0.9244 ± 0.0093
	l-b-l	once	0.7438 ± 0.0113	**0.7199 ± 0.0099**	0.8077 ± 0.0260	0.9087 ± 0.0045	0.9205 ± 0.0022
		each	0.7457 ± 0.0008	0.7076 ± 0.0423	0.8046 ± 0.0249	**0.9103 ± 0.0052**	**0.9281 ± 0.0067**

Analysis of simplified graphs. In this section, we discuss how the simplification impacts the structural characteristics of the multilayer graphs. For each dataset, we compare structural characteristics before and after the simplification with TAMARA is performed. We show results for one of the prediction sub-tasks, user migration prediction on *um-econ* (Table 3) while the others can be consulted in the Appendix. It can be noted how the impact of TAMARA(NS) can be different on each layer of a specific graph, while the action of TAMARA(DE) seems to be more uniform over a given graph. Once again, this is consistent with the fact that we are comparing a task-aware and a random approach. The clearest impact is observed on the number of intra-edges, TAMARA drastically reduces the number of edges while still improving the performance: this is extremely important as the computation cost of graph convolution is linear in the number of graph edges [12], making a reduced number edges an ideal property. In addition, some interesting observations can be drawn about label assortativity, i.e., the similarity of connections in the graph with respect to node labels (high label assortativity means that a node is more likely to connect with a node with the same label). We can see how TAMARA(NS) tends to increase label assortativity across layers: this makes sense as TAMARA(NS) can leverage node features, so it would be able to preserve the connections between similar nodes. Such behavior

cannot be replicated by the random procedure behind TAMARA(DE). Similarly, as regards transitivity (i.e., the fraction of all possible triangles present in a graph), we can observe a general decrease, since the number of triangles is necessarily reduced as we remove edges. However, on layers with lower transitivity (< 0.1), only TAMARA(NS) increases transitivity values: this can be observed in *um-econ* and *um-socioeco* (Supporting Information).

The relevance of training jointly simplification and graph neural network is, therefore, the most important observation: during the training, TAMARA improves its capacity to recognize edges that are unrelated to the task at hand, allowing it to determine which graph characteristics are most crucial to maintain or enhance. Additionally, TAMARA demonstrates the capability of significantly reducing the number of edges while improving or at least keeping performance.

Table 3: Statistics for each graph layer before and after the simplification on um-econ dataset.

	ℓ	Intra edges	Label assortativity	Transitivity	Indegree mean	Indegree max	Outdegree mean	Outdegree max
TAMARA (NS)	L0	174381.00	0.08	0.01	23.63	3610.00	23.63	6021.00
		6207.00	0.35	0.02	1.17	328.00	1.02	3.00
		(-96.44%)	(+320.57%)	(+86.70%)	(-95.06%)	(-90.91%)	(-95.69%)	(-99.95%)
	L1	35060.00	0.27	0.00	5.55	937.00	5.55	4769.00
		5038.00	0.62	0.02	0.87	145.00	1.02	3.00
		(-85.63%)	(+127.57%)	(+2947.23%)	(-84.39%)	(-84.53%)	(-81.71%)	(-99.94%)
TAMARA (DE)	L0	174381.00	0.08	0.01	23.63	3610.00	23.63	6021.00
		121999.00	0.08	0.01	16.53	2552.00	16.53	4230.00
		(-30.04%)	(+2.17%)	(-31.27%)	(-30.03%)	(-29.31%)	(-30.03%)	(-29.75%)
	L1	35060.00	0.27	0.00	5.55	937.00	5.55	4769.00
		24578.00	0.27	0.00	3.89	642.00	3.89	3349.00
		(-29.90%)	(-1.44%)	(-31.29%)	(-29.87%)	(-31.48%)	(-29.89%)	(-29.78%)

Hyperparameters sensitivity analysis. As a last analysis step, we study the impact of varying the main hyperparameters, i.e., the drop rate p for TAMARA(DE) and k for TAMARA(NS). In Figure 2 we report a sensitivity analysis for p and k, where the other hyperparameters are set to the best performing combination. We can see that for *um-econ*, low k values lead to lower performance. Similarly, a high drop rate p seems to lead to worse performance. For *imdb-mlh*, we can draw similar observations for p (i.e., a high drop worsens the performance), while variations of k seem to have a minor impact on the process. Similarly, the impact of k is minor also for *Koumbia-2*. In this case, the impact of p seems to be reduced too.

Overall, the takeaway is that both TAMARA(NS) and TAMARA(DE) are not very sensitive to variations of their respective main hyperparameters k and p. This makes their use more solid and easy, by making hyperparameter tuning relatively unimportant.

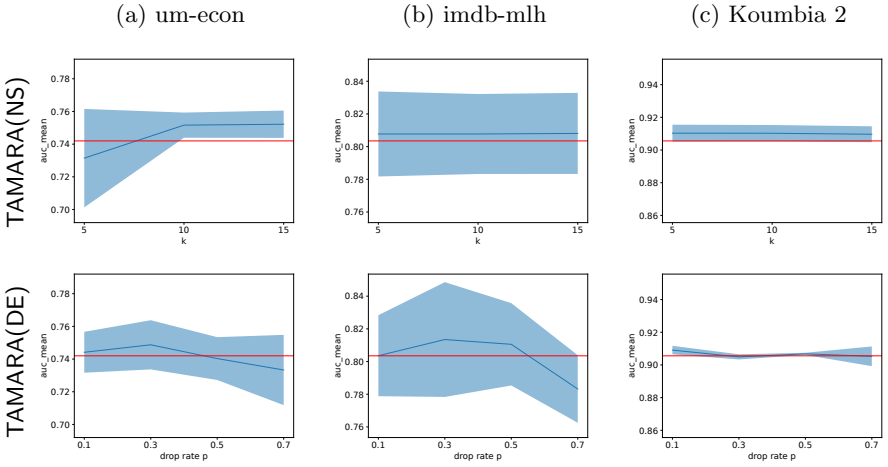


Fig. 2: Sensitivity analysis based on AUC, for the 2-layer graphs. We compare the AUC for the baseline (in red), with the AUC (average, standard deviation) for the best simplification method (in blue). The remaining ones can be found in the appendix.

6 Conclusions

The findings presented in this work show the significance of the proposed framework: TAMARA leads to significant performance improvements, by selecting the best available simplification strategies. These advances in performance are even more noteworthy when we take into account that TAMARA achieves them while drastically reducing the number of edges. Most importantly, TAMARA shows the importance of training jointly for the simplification and node classification tasks: as the ability to identify task-irrelevant edges increases, TAMARA is guided in discovering the most important graph properties to preserve or enhance.

Future research will focus on analyzing how multilayer simplification can be beneficial for a variety of tasks, including link prediction (removing unimportant or "spam" links to improve prediction performance), clustering (removing redundant links should improve boundaries between clusters, thus improving cluster quality), and graph classification (removing noisy links should help in the identification of similar graphs). Finally, additional future works will focus on the interaction between graph properties and downstream tasks to support multilayer simplification. A better understanding of graph properties can be beneficial in the development of simplification algorithms and overall it could lead to a better understanding of complex systems in different domains.

References

1. Arenas, A., Díaz-Guilera, A., Kurths, J., Moreno, Y., Zhou, C.: Synchronization in complex networks. Physics reports **469**(3), 93–153 (2008)
2. Ba, C.T., Michienzi, A., Guidi, B., Zignani, M., Ricci, L., Gaito, S.: Fork-based user migration in blockchain online social media. In: 14th ACM Web Science Conference 2022. pp. 174–184 (2022)
3. Ba, C.T., Zignani, M., Gaito, S.: The role of cryptocurrency in the dynamics of blockchain-based social networks: The case of steemit. PloS one **17**(6), e0267612 (2022)
4. Battiston, S., Caldarelli, G., D'Errico, M.: The financial system as a nexus of interconnected networks. In: Interconnected networks, pp. 195–229. Springer (2016)
5. Costa, L.D.F., Oliveira, Osvaldo, J., Travieso, G., Rodrigues, F.A., Villas Boas, P., Antiqueira, L., Viana, M.P., Correa Rocha, L.: Analyzing and modeling real-world phenomena with complex networks: a survey of applications. Advances in Physics **60**(3), 329–412 (May 2011)
6. Curzel, J.L., Lüders, R., Fonseca, K.V.O., Rosa, M.: Temporal performance analysis of bus transportation using link streams. Mathematical Problems in Engineering (2019)
7. Dickison, M.E., Magnani, M., Rossi, L.: Multilayer social networks. Cambridge University Press (2016)
8. Hamilton, W., Ying, Z., Leskovec, J.: Inductive representation learning on large graphs. Advances in neural information processing systems **30** (2017)
9. Interdonato, R., Gaetano, R., Seen, D.L., Roche, M., Scarpa, G.: Extracting multilayer networks from sentinel-2 satellite image time series. Network Science **8**(S1), S26–S42 (2020)
10. Interdonato, R., Magnani, M., Perna, D., Tagarelli, A., Vega, D.: Multilayer network simplification: approaches, models and methods. Comput. Sci. Rev. **36**, 100246 (2020)
11. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
12. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
13. Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J.P., Moreno, Y., Porter, M.A.: Multilayer networks. Journal of complex networks **2**(3), 203–271 (2014)
14. Li, D., Yang, T., Du, L., He, Z., Jiang, L.: Adaptivegcn: Efficient gcn through adaptively sparsifying graphs. Proceedings of the 30th ACM International Conference on Information & Knowledge Management (2021)
15. Liu, Y., Safavi, T., Dighe, A., Koutra, D.: Graph summarization methods and applications: A survey. ACM computing surveys (CSUR) **51**(3), 1–34 (2018)
16. Luo, D., Cheng, W., Yu, W., Zong, B., Ni, J., Chen, H., Zhang, X.: Learning to drop: Robust graph neural network via topological denoising. In: Proceedings of the 14th ACM International Conference on Web Search and Data Mining. pp. 779–787 (2021)
17. Martirano, L., Zangari, L., Tagarelli, A.: Co-mlhan: contrastive learning for multilayer heterogeneous attributed networks. Applied Network Science **7**(1), 1–44 (2022)
18. Nekovee, M., Moreno, Y., Bianconi, G., Marsili, M.: Theory of rumour spreading in complex social networks. Physica A: Statistical Mechanics and its Applications **374**(1), 457–470 (2007)

19. Pastor-Satorras, R., Castellano, C., Van Mieghem, P., Vespignani, A.: Epidemic processes in complex networks. Rev. Mod. Phys. **87**, 925–979 (Aug 2015). https://doi.org/10.1103/RevModPhys.87.925, https://link.aps.org/doi/10.1103/RevModPhys.87.925
20. Rong, Y., Huang, W., Xu, T., Huang, J.: Dropedge: Towards deep graph convolutional networks on node classification. In: ICLR (2020)
21. Shanthamallu, U.S., Thiagarajan, J.J., Song, H., Spanias, A.: Gramme: Semisupervised learning using multilayered graph attention models. IEEE Trans. Neural Networks Learn. Syst. **31**(10), 3977–3988 (2020). https://doi.org/10.1109/TNNLS.2019.2948797, https://doi.org/10.1109/TNNLS.2019.2948797
22. Thompson, W., Brantefors, P., Fransson, P.: From static to temporal network theory: Applications to functional brain connectivity. Network Neuroscience **1**, 1–56 (04 2017). https://doi.org/10.1162/NETN_a_00011
23. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
24. Wang, L., Yu, W., Wang, W., Cheng, W., Zhang, W., Zha, H., feng He, X., Chen, H.: Learning robust representations with graph denoising policy network. 2019 IEEE International Conference on Data Mining (ICDM) pp. 1378–1383 (2019)
25. Wickman, R., Zhang, X., Li, W.: Sparrl: Graph sparsification via deep reinforcement learning. ArXiv **abs/2112.01565** (2021)
26. You, J., Du, T., Leskovec, J.: Roland: graph learning framework for dynamic graphs. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 2358–2366 (2022)
27. You, J., Ying, R., Leskovec, J.: Position-aware graph neural networks. In: International conference on machine learning. pp. 7134–7143. PMLR (2019)
28. Zangari, L., Interdonato, R., Calió, A., Tagarelli, A.: Graph convolutional and attention models for entity classification in multilayer networks. Applied Network Science **6**(1), 1–36 (2021)
29. Zhang, M., Chen, Y.: Link prediction based on graph neural networks. Advances in neural information processing systems **31** (2018)
30. Zhang, Z., Bu, J., Ester, M., Zhang, J., Yao, C., Yu, Z., Wang, C.: Hierarchical graph pooling with structure learning. arXiv preprint arXiv:1911.05954 (2019)
31. Zhao, L., Wang, G.J., Wang, M., Bao, W., Li, W., Stanley, H.E.: Stock market as temporal network. Physica A: Statistical Mechanics and its Applications **506**, 1104–1112 (Sep 2018). https://doi.org/10.1016/j.physa.2018.05.039, http://dx.doi.org/10.1016/j.physa.2018.05.039
32. Zheng, C., Zong, B., Cheng, W., Song, D., Ni, J., Yu, W., Chen, H., Wang, W.: Robust graph representation learning via neural sparsification. In: ICML (2020)

A Appendix

Notation table

Table 4: Summary of notations used in the paper and their description.

Notations	Description
$\mathcal{G}_{\mathcal{L}}$	Multilayer graph
$\mathcal{V}$	Set of N entities (e.g., users)
$\mathcal{L}, \ell, L_l$	Set of layers, number of layers, l-th layer
$\mathcal{V}_{\mathcal{L}}$	Set of nodes in $\mathcal{G}_{\mathcal{L}}$
$\mathcal{E}_{\mathcal{L}}$	Set of edges $\mathcal{G}_{\mathcal{L}}$
A, A_ℓ	Adjacency matrix in G, Adjacency matrix of the l-th layer of $\mathcal{G}_{\mathcal{L}}$
A^{sup}	Supra-adjacency matrix
$\widetilde{A}, \widetilde{A}^{sup}$	Adjacency matrix and supra-adjacency matrix with self loops
v_i, i	Index i of a node $V_i \in \mathcal{V}_{\mathcal{L}}$
$\Gamma(i)$	Neighborhood of node V_i
$\Gamma(i,l)$	Within-layer neighborhood of node V_i
$\Psi(i,l)$	Outside-layer neighborhood of node V_i
X, X_l	Attribute (input feature) matrix, resp. in the l-th layer of $\mathcal{G}_{\mathcal{L}}$
$x, x_{(i,l)}$	Attribute (input feature) vector for node v_i, resp. node v_i in the l-th layer of $\mathcal{G}_{\mathcal{L}}$
f	Number of attributes (input features)
E	Edge attribute matrix
f_E	Number of edge attributes
$\mathcal{G}_{(\mathcal{L},\mathcal{X},\mathcal{E})}$	Attributed multilayer graph
d	Size of the embedding
Z, Z_l	Embedding (output feature) matrix, resp. in the l-th layer of $\mathcal{G}_{\mathcal{L}}$
$z_i, z_{(i,l)}$	Embedding (output feature) vector for node v_i, resp. node v_i in the l-th layer of $\mathcal{G}_{\mathcal{L}}$
W, W^k	Weight matrix of a generic, resp. weights of l-th GNN layers
$f_W, f_W^{(k)}$	GNN module, GNN at the k-th GNN layers
K, k	Number of GNN layers, index of a layer of the GNN
$H^{(k+1)} = f_W^{(k)}(H^{(k)}, A)$	A GNN layer computation
f_{θ_S}, $f_{\theta_S^k}^k$	simplification neural network and its parameters, resp. simplification neural network for a certain GNN layer
h_i	Hidden layer vector for node v_i
$h_{(i,l)}^{(k)}$	Hidden layer vector at the k-th layer of the GNN for entity v_i in layer L_l of $\mathcal{G}_{\mathcal{L}}$
Y, $\hat{Y}$	Ground truth, predictions

Analysis of simplified graphs - datasets not included in the paper

Table 5: Statistics for each graph layer before and after the simplification on imdb-mlh dataset.

	ℓ	Intra edges	Label assortativity	Transitivity	Indegree mean	Indegree max	Outdegree mean	Outdegree max
TAMARA (NS)		6121.00	0.70	0.40	4.27	79.00	4.27	79.00
	L0	2818.00	0.87	0.29	3.09	42.00	3.09	40.00
		(-53.96%)	(+23.27%)	(-28.36%)	(-27.55%)	(-46.84%)	(-27.55%)	(-49.37%)
		5355.00	0.72	0.38	4.00	69.00	4.00	69.00
	L1	2816.00	0.90	0.00	3.09	42.00	3.09	38.00
		(-47.41%)	(+24.60%)	(-100.00%)	(-22.63%)	(-39.13%)	(-22.63%)	(-44.93%)
TAMARA (DE)		6121.00	0.70	0.40	4.27	79.00	4.27	79.00
	L0	4277.00	0.71	0.26	3.00	55.00	2.98	53.00
		(-30.13%)	(+1.44%)	(-34.14%)	(-29.80%)	(-30.38%)	(-30.27%)	(-32.91%)
		5355.00	0.72	0.38	4.00	69.00	4.00	69.00
	L1	3749.00	0.73	0.26	2.79	46.00	2.81	49.00
		(-29.99%)	(+0.47%)	(-29.83%)	(-30.22%)	(-33.33%)	(-29.71%)	(-28.99%)

Table 6: Statistics for each graph layer before and after the simplification on Koumbia 2 dataset.

	ℓ	Intra edges	Label assortativity	Transitivity	Indeg mean	Indegree max	Outdegree mean	Outdegree max
TAMARA (NS)		5724.00	0.72	0.16	4.39	20.00	4.39	24.00
	L0	2254.00	0.90	0.00	2.85	11.00	2.85	11.00
		(-60.62%)	(+24.26%)	(-100.00%)	(-35.18%)	(-45.00%)	(-35.18%)	(-54.17%)
		4779.00	0.79	0.20	3.97	25.00	3.97	27.00
	L1	2253.00	0.91	0.00	2.85	22.00	2.85	20.00
		(-52.86%)	(+15.07%)	(-100.00%)	(-28.32%)	(-12.00%)	(-28.32%)	(-25.93%)
TAMARA (DE)		5724.00	0.72	0.16	4.39	20.00	4.39	24.00
	L0	2909.00	0.72	0.08	2.21	13.00	2.22	15.00
		(-49.18%)	(-0.67%)	(-52.59%)	(-49.64%)	(-35.00%)	(-49.55%)	(-37.50%)
		4779.00	0.79	0.20	3.97	25.00	3.97	27.00
	L1	2356.00	0.79	0.09	1.97	13.00	1.97	17.00
		(-50.70%)	(+0.24%)	(-53.31%)	(-50.41%)	(-48.00%)	(-50.50%)	(-37.04%)

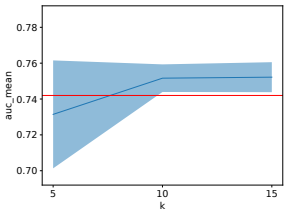
Table 7: Statistics for each graph layer before and after the simplification on um-socioeco dataset.

		Intra edges	Label Assortativity	Transitivity	Indegree mean	Indegree max	Outdegree mean	Outdegree max
TAMARA (NS)	L0	579352.00	0.06	0.20	130.25	1875.00	130.25	2990.00
		4624.00	0.14	0.00	4.02	31.00	4.02	5.00
		(-99.20%)	(+146.19%)	(-100.00%)	(-96.92%)	(-98.35%)	(-96.92%)	(-99.83%)
	L1	476439.00	0.05	0.11	107.64	1759.00	107.64	4262.00
		4652.00	0.15	0.01	4.02	34.00	4.02	6.00
		(-99.02%)	(+181.36%)	(-95.09%)	(-96.26%)	(-98.07%)	(-96.26%)	(-99.86%)
	L2	74580.00	0.12	0.01	19.38	2543.00	19.38	3753.00
		4603.00	0.44	0.03	4.01	331.00	4.01	5.00
		(-93.83%)	(+277.44%)	(+287.20%)	(-79.30%)	(-86.98%)	(-79.30%)	(-99.87%)
	L3	14856.00	0.39	0.01	6.26	276.00	6.26	701.00
		4586.00	0.73	0.00	4.01	66.00	4.01	5.00
		(-69.13%)	(+85.09%)	(-100.00%)	(-36.02%)	(-76.09%)	(-36.02%)	(-99.29%)
TAMARA (DE)	L0	579352.00	0.06	0.20	130.25	1875.00	130.25	2990.00
		492450.00	0.06	0.17	111.16	1601.00	111.16	2543.00
		(-15.00%)	(+0.84%)	(-15.26%)	(-14.65%)	(-14.61%)	(-14.65%)	(-14.95%)
	L1	476439.00	0.05	0.11	107.64	1759.00	107.64	4262.00
		404974.00	0.05	0.09	91.95	1493.00	91.95	3623.00
		(-15.00%)	(+0.57%)	(-15.12%)	(-14.58%)	(-15.12%)	(-14.58%)	(-14.99%)
	L2	74580.00	0.12	0.01	19.38	2543.00	19.38	3753.00
		63393.00	0.12	0.01	16.92	2173.00	16.92	3201.00
		(-15.00%)	(-0.89%)	(-14.71%)	(-12.68%)	(-14.55%)	(-12.68%)	(-14.71%)
	L3	14856.00	0.39	0.01	6.26	276.00	6.26	701.00
		12628.00	0.39	0.01	5.77	229.00	5.77	604.00
		(-15.00%)	(+0.24%)	(-14.64%)	(-7.81%)	(-17.03%)	(-7.81%)	(-13.84%)

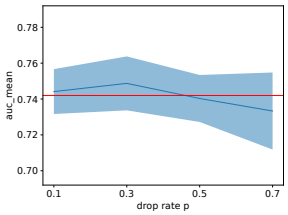
Table 8: Statitistics for each graph layer before and after the simplification on Koumbia 5 dataset.

		Intra edges	Label Assortativity	Transitivity	Indegree mean	Indegree max	Outdegree mean	Outdegree max
TAMARA (NS)		4157.00	0.84	0.25	7.15	33.00	7.15	38.00
	L0	2252.00	0.95	0.00	6.30	28.00	6.30	28.00
		(-45.83%)	(+12.85%)	(-100.00%)	(-11.87%)	(-15.15%)	(-11.87%)	(-26.32%)
		5752.00	0.70	0.22	9.03	39.00	9.03	36.00
	L1	2249.00	0.90	0.00	7.47	27.00	7.47	25.00
		(-60.90%)	(+28.24%)	(-100.00%)	(-17.27%)	(-30.77%)	(-17.27%)	(-30.56%)
		4951.00	0.70	0.22	8.64	47.00	8.64	53.00
	L2	2252.00	0.88	0.00	7.44	41.00	7.44	41.00
		(-54.51%)	(+25.94%)	(-100.00%)	(-13.91%)	(-12.77%)	(-13.91%)	(-22.64%)
		3635.00	0.98	0.24	6.82	39.00	6.82	42.00
	L3	2252.00	1.00	0.00	6.21	37.00	6.21	36.00
		(-38.05%)	(+1.36%)	(-100.00%)	(-9.02%)	(-5.13%)	(-9.02%)	(-14.29%)
		5605.00	0.68	0.20	9.29	48.00	9.29	50.00
	L4	2266.00	0.87	0.04	7.80	41.00	7.80	41.00
		(-59.57%)	(+28.12%)	(-79.11%)	(-16.00%)	(-14.58%)	(-16.00%)	(-18.00%)
TAMARA (DE)		4157.00	0.84	0.25	7.15	33.00	7.15	38.00
	L0	3534.00	0.85	0.20	6.87	33.00	6.87	37.00
		(-14.99%)	(+0.65%)	(-18.32%)	(-3.88%)	(-)	(-3.88%)	(-2.63%)
		5752.00	0.70	0.22	9.03	39.00	9.03	36.00
	L1	4890.00	0.70	0.19	8.65	36.00	8.65	35.00
		(-14.99%)	(+0.28%)	(-16.06%)	(-4.25%)	(-7.69%)	(-4.25%)	(-2.78%)
		4951.00	0.70	0.22	8.64	47.00	8.64	53.00
	L2	4209.00	0.69	0.19	8.31	46.00	8.31	52.00
		(-14.99%)	(-1.12%)	(-13.45%)	(-3.82%)	(-2.13%)	(-3.82%)	(-1.89%)
		3635.00	0.98	0.24	6.82	39.00	6.82	42.00
	L3	3090.00	0.98	0.20	6.58	39.00	6.58	40.00
		(-14.99%)	(+0.18%)	(-14.39%)	(-3.56%)	(-)	(-3.56%)	(-4.76%)
		5605.00	0.68	0.20	9.29	48.00	9.29	50.00
	L4	4765.00	0.68	0.17	8.92	47.00	8.92	48.00
		(-14.99%)	(-0.38%)	(-14.34%)	(-4.03%)	(-2.08%)	(-4.03%)	(-4.00%)

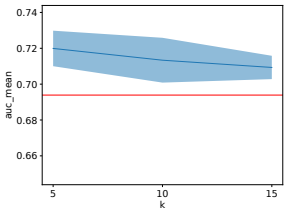
Hyperparameters sensitivity analysis - all datasets



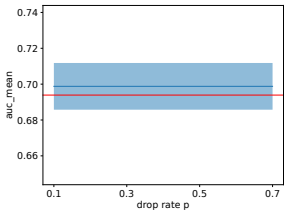
(a) um-econ-TAMARA(NS)



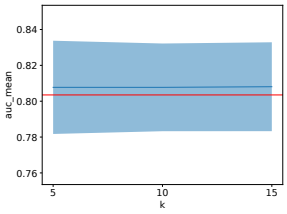
(b) um-econ-TAMARA(DE)



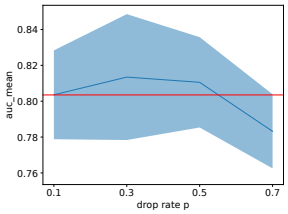
(c) um-socioeco-TAMARA(NS)



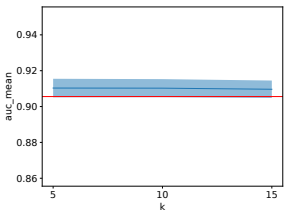
(d) um-socioeco-TAMARA(DE)



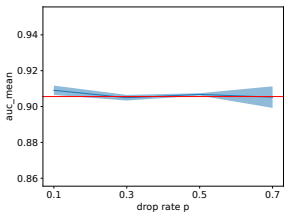
(e) imdb-mlh-TAMARA(NS)



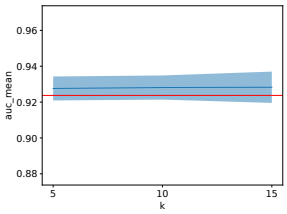
(f) imdb-mlh-TAMARA(DE)



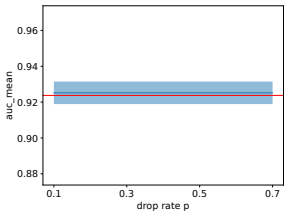
(g) Koumbia 2-TAMARA(NS)



(h) Koumbia 2-TAMARA(DE)



(i) Koumbia 5-TAMARA(NS)



(j) Koumbia 5-TAMARA(DE)

Fig. 3: Sensitivity test auc.

Hyperparameter tuning - parameter space

```
{'datasets': [('um-econ', 'features'),
  ('um-socioeco', 'features'),
  ('imdb-mlh', 'features'),
  ('Koumbia_2', 'features'),
  ('Koumbia_5', 'features')],
 'architecture': ['multi'],
 'architecture_simp': ['multi', 'single'],
 'model': ['gcn', 'gcn-de', 'gcn-ns'],
 'gnn_level': [True, False],
 'drop_rate_p': [0.1, 0.3, 0.5, 0.7],
 'k': [5, 10, 15],
 'tau': [0.001],
 'standardize': [True],
 'feat-variability': ['fixed'],
 'split': ['25 50 25'],
 'plots': [True],
 'early-stop': [True],
 'fastmode': [True],
 'gpu': [1],
 'run': [1],
 'debugging': [False],
 'dropout': [0.3],
 'hidden': [16, 32],
 'lr': [0.002],
 'num-layers': [2],
 'ns_num_hidden': [32],
 'epochs': [250],
 'patience': [10]}
```