Performance of Community Detection Algorithms Supported by Node Embeddings

Bartosz Pankratz* Bogumił Kamiński† Paweł Prałat‡

May 12, 2024

Abstract

Grouping of nodes into subsets, that are relatively densely interconnected and separable from the rest of the network is a property often displayed in many real-world complex networks; this feature is known as a community structure. There is a growing demand for algorithms, that can find partition resembling the community structure of a given network as closely as possible. However, most popular algorithms for community detection in graphs have one serious drawback, namely, they are heuristic-based and in many cases are unable to find a near-optimal solution. Moreover, their results are volatile, impacting the replicability of their results.

In this paper, we investigate if the performance of greedy algorithms might be improved by initialization of such algorithms with some carefully chosen partition of nodes, namely a partition obtained by embedding the nodes into real numbers space and then running a clustering algorithm on this latent representation. We believe that embedding will filter unwanted noise while retaining the proximity of nodes belonging to the same community or will learn more complex and elusive relations between nodes. Then, clustering algorithms run on this embedding will create a stable partitioning that will reduce the uncertainty in the initial phases of the community detection algorithms.

The experiments show that the proposed procedure significantly improves the results over baseline community detection algorithms, namely **Louvain** and **Leiden**. It also reduces the inherent volatility of such algorithms. The impact depends on the given graph's properties, especially the strength of the community structure and degree distribution. The largest boost in performance is given in the cases when networks are "noisier", that is when the community structure is less pronounced and there are many connections between communities. Furthermore, the design and parametrization of the procedure depend on the network's topology, not on the size of the network itself.

1 Introduction

Empirical complex networks tend to display a modular organization which means that one can separate sets of nodes (inducing *subgraphs*) with a considerably larger density of edges between nodes

*Decision Analysis and Support Unit, SGH Warsaw School of Economics, Warsaw, Poland; e-mail: bpankra@sgh.waw.pl

†Decision Analysis and Support Unit, SGH Warsaw School of Economics, Warsaw, Poland; e-mail: bogumil.kaminski@sgh.waw.pl

‡Department of Mathematics, Toronto Metropolitan University, Toronto, ON, Canada; e-mail: pralat@torontomu.ca

in such sets than between two different sets. Such a property is widely referred to as a *community structure* [12]. In social networks, communities may represent groups by interest. For example in citation networks, they correspond to related papers. In Web communities, they are formed by pages on related topics, etc. Being able to identify communities in a network could help us utilize this network more effectively and learn much more interesting and non-trivial relations between nodes. Indeed, with growing awareness of the network-like nature of many phenomena in the surrounding world, community detection is productively used in many different scenarios: identification of interest groups in social networks [5, 10, 41], frauds in telecommunications networks [35, 37], or various biomedical applications [17, 45].

For a graph $G = (V, E)$, with $n = |V|$ nodes and $m = |E|$ edges, we define each community in G as a subset C of V that induces a subgraph $G[C]$. Obviously, we expect that $G[C]$ is connected but, more importantly, nodes in a community should be more likely to be connected to other members of the same community than to nodes in other communities.

Partition $\mathbf{P} = \{P_1, P_2, \ldots, P_\ell\}$ of the set V is a grouping of its elements into non-empty subsets in such a way that every element is included in exactly one subset and subsets are pairwise disjoint. The goal of community detection process is to find a partition that captures the structure of communities in a graph. However, we should ask a very important question: *how can one asses that the partition* $\mathbf{P}$ *represents the underlying community structure in* G*?*

Communities are somewhat elusive; without full knowledge about a graph generating process (which is obviously the case for most real-world networks) it is not clear what score function or measure should be used to assess them and, consequently, what algorithm should be used to detect them, especially since no algorithm can uniquely solve community detection task [33]. This problem is widely discussed, see, for example, [26, 50, 28], and plenty of different score functions were proposed up to them. The *modularity* function [30] is possibly the most often used one.

The Modularity function measures the difference between the number of edges within groups induced by a given partition $\mathbf{P}$ and the expected number of such edges if they were attached randomly, regardless of community structure. The expected number of edges is given by an appropriately selected *null-model*, usually **Chung-Lu** random graph model [1].

For a graph $G = (V, E)$, a given partition $\mathbf{P} = \{P_1, P_2, \ldots, P_\ell\}$ of V and **Chung-Lu** null-model, the modularity function is defined as follows:

$$q_G(\mathbf{P}) \quad = \quad \sum_{P_i \in \mathbf{P}} \frac{e_G(P_i)}{|E|} - \sum_{P_i \in \mathbf{P}} \left(\frac{\mathrm{vol}(P_i)}{\mathrm{vol}(V)} \right)^2, \tag{1}$$

The first term in (1), $\sum_{P_i \in \mathbf{P}} e(P_i)/|E|$, is called the *edge contribution*, and it computes the fraction of edges that fall within one of the parts. The second one, $\sum_{P_i \in \mathbf{P}} (\mathrm{vol}(P_i)/\mathrm{vol}(V))^2$, is called the *degree tax*, and it computes the expected fraction of edges that do the same in the corresponding random graph (the null-model). The intuition behind the modularity function is straightforward; it measures how much the community structure in G differs from the one in the purely random graph, where we expect no community structure. Obviously, a higher value of modularity indicates a stronger division of a network. The natural simplicity of the definition of modularity is its biggest advantage; it is, at the same time, a perfect candidate for global criterion to define communities, a quality function of community detection algorithms, and a way to measure the presence of community structure in a network, without introducing any specific centrality of (dis)similarity measures for nodes.

Finding an optimal partition $\mathbf{P}$ of graph G is a non-trivial problem. Solving it directly, by enumerating over all partitions of the set of nodes V until the maximum modularity $q^*(G)$ is

reached, is practically impossible. Even for a small graph, with e.g. 15 nodes, such a procedure will be computationally too expensive to conduct. Moreover, optimizing modularity is a NP-hard problem [7]. There exist exhaustive optimization methods (for example, via **the Bayan algorithm** [2] or **simulated annealing** [16]), but the complexity of the problem limits their usability for large graphs. As a result, to partition the graph one must rely on heuristic-based, greedy algorithms.

One of the most popular, fastest, and best performing [25] one is the **Louvain** algorithm [6]. Its core idea is simple yet effective. It is a two-step technique: at first, it locally optimizes the modularity in the neighbourhood of each node v_i. It considers all neighbours of v_i and moves it to the community that provides the largest increase of the score function. During the second step, when all nodes are properly placed, it aggregates the communities into super-nodes, where all edges between two communities are replaced by a single weighted edge. Then, both phases are repeated until no improvement of the score function can be further achieved. By default, the **Louvain** algorithm starts from a singleton partition in which each node belongs to its own community. However, it is possible to initialize the algorithm with a preexisting partitioning.

Louvain algorithm offers a great compromise between the accuracy of the estimate of the modularity maximum, which is better than that delivered by the greedy techniques, and efficiency—the run time is only $\mathrm{O}(|V| \ln^2 |V|)$. But it has some severe and known drawbacks.

First, the obtained results are heavily stochastic, that is, each run of the algorithm on the same network may lead to vastly different partitions. Moreover, it may create weakly connected or even internally disconnected communities [44]. These problems are caused by two factors, both inherent to the nature of the algorithm. It is a greedy algorithm; sometimes, especially on early iterations, nodes might be added to communities they should not belong to because the algorithm finds the local best solution without considering the broader structure of the graph. Then, during the second phase, it merges the community into a supernode making it impossible to backtrack and fix these bad early connections.

This shortcoming might be addressed in two manners; either by allowing the algorithm to backtrack and refine the created communities in each step, which was proposed in the **Leiden** algorithm [44], or by ensuring that the initial partitioning is stable and contains the nodes that certainly belong to the same community, as in **ECG** (Ensemble Clustering algorithm for Graphs), a consensus clustering algorithm [36]. At the beginning, it generates k independent initial partitions: $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_k$ based on different random permutations of nodes. Each partition is created by running the first iteration(s) of **Louvain**. Then **ECG** aggregates outputs into G_w, a weighted modification of graph G, where the weight of an edge $v_i v_j$ belonging to the 2-core of G depends on the number of co-appearances in the previously obtained partitions. Finally, the **Louvain** algorithm is run on G_w.

1.1 Contributions

In the conference version of this paper [32], we proposed a new, three-step method of community detection with initial partitioning obtained on the embedded representation of a graph. The procedure starts with embedding the graph nodes in the space of real numbers. Then, clustering algorithm is run on the data. The algorithm is fine-tuned to obtain a stable partition which is finally used to initialize the **Louvain** or **Leiden** algorithms. In this paper, we use the **EC** prefix (**EC** stands for **Embedding-Clustering**) to denote the proposed extension of **Louvain** or **Leiden** algorithms.

The reasoning behind this approach is as follows: carefully selected embeddings preserve the proximity of nodes belonging to the same community and clearly separate them from the other

ones, reducing the chance of misguided connections at the early stages of the algorithm. However, relying only on the embedded representation causes problems on its own; by their nature (typically local), embeddings preserve some properties of the nodes but filter some other ones, resulting in the inherent information loss that might induce a significant bias if we decide to run the clustering algorithm only on the embedded data and use it as the final partition. Therefore, the most promising approach is to combine both methods.

Initial experiments show that, indeed, this method is capable of improving the performance of both, **Louvain** and **Leiden** algorithms. However, rerunning the presented procedure for every graph is tedious and impractical. Thus, it is necessary to deepen the analysis and find patterns that will serve as guidance for potential users of this combined approach.

The goal of this paper is to find such regularities in data and better understand the reasons behind the performance of the presented method. To achieve that, we run far more complex experiments using the artificial networks generated by the **ABCD** (**A**rtificial **B**enchmark for **C**ommunity **D**etection) model [23, 20] and perform various tests on the obtained results. Using artificial networks, we could try out the proposed method in various, yet controllable, environments and better understand the relations between the properties of the network and the performance of the additional clustering step. Moreover, to ensure that results are valid for different networks, we run similar experiments on datasets containing various mutual like networks among verified Facebook pages [40].

2 Method Description

Let $G = (V, E)$ be a graph on the set of n nodes $V = \{v_1, v_2, \ldots, v_n\}$ and the set of m edges $E = \{e_1, e_2, \ldots, e_m\}$. In order to find the partition $\mathbf{P} = \{P_1, P_2, \ldots, P_\ell\}$ of V that tries to maximize the modularity function $q_G(\mathbf{P})$, we perform the following three steps:

> **Step 1:** Find an embedding function $\mathcal{E}\colon V \to \mathbb{R}^s$ which embeds each node of graph G into a s-dimensional vector $\mathcal{E}(v) = \{z_1, z_2, \ldots, z_s\}$, where $s \ll n$.
>
> **Step 2:** Run the clustering algorithm on the obtained latent representation $\mathcal{E}$ to get partition $\mathbf{A} = \{A_1, A_2, \ldots, A_k\}$. The goal is to use $\mathbf{A}$ as an initializing partitioning for the **Louvain** (or **Leiden**) algorithm, so the number of clusters k should be significantly larger than the desired number of parts in the partition $\mathbf{P}$: $k \gg \ell$.
>
> **Step 3:** Run the **Louvain** (or **Leiden**) algorithm on graph G using partition $\mathbf{A}$ as a starting point. The result of this procedure, partition $\mathbf{P}$, is the outcome of our algorithm.

Figure 1 summarizes the proposed method.

One of the main reasons why one might want to use the embeddings is the nature of graphs as data structures; they are discrete objects. As a result, the number of possible approaches to community detection problem is far smaller than in the case of clustering of real numbers data. It forces one to use heuristic-based approaches such as the classical **Louvain** algorithm. On the other hand, embedded latent representation is a vector of real numbers, which creates new possibilities. Mainly because there are more algorithms designed for working with real numbers and they are often more efficient [3]. Also, a properly selected embedding might be considered a form of "denoising" data. It retains only the properties of nodes that are important for the task at hand, removing the remaining useless relations, resulting in a representation of data that is significantly lower dimensional and possibly easier to cluster.

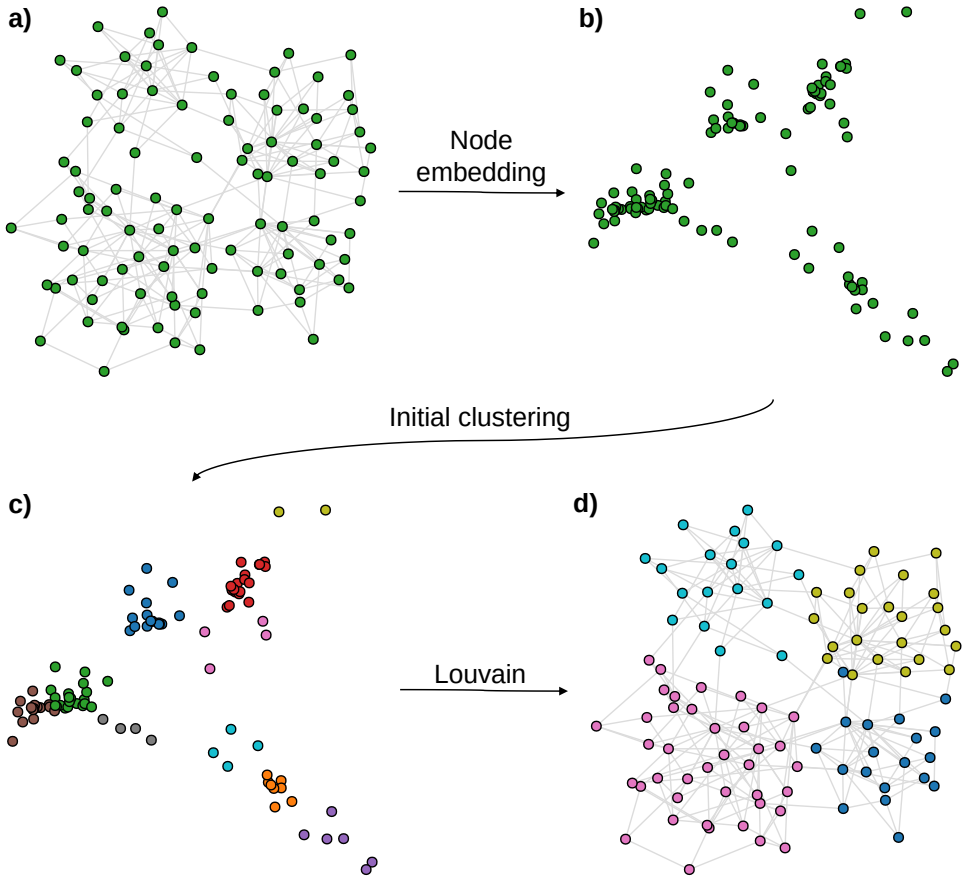


Figure 1: Schematic of the proposed method. Graph G (**a**) is embedded into latent space (**b**). Then, the clustering is performed on the embedded data (**c**). Obtained clusters are used as a starting point for the **Louvain** (or **Leiden**) algorithm (**d**).

However, experimental results [42] show that using only an embedded representation to obtain the desired partition is not enough. Embeddings are usually too reductive, and the representation gap between graph G and its latent representation $\mathcal{E}$ is too large. Indeed, embeddings preserve some proximity of nodes but remove other useful global information that might be crucial to achieving a satisfactory result. Overcoming this issue is the main reason why the proposed solution consists of two separate partitioning steps. The reasoning is as follows: starting the **Louvain** with a visibly smaller starting set of nodes in which most sensitive elements are already connected should improve the results and decrease the volatility of the method.

Moreover, such a combined method has one more important advantage over traditional greedy algorithms; it combines different definitions of communities, giving a more complex and refined perspective on the task. Greedy algorithms, such as **Louvain**, optimize a score function (usually modularity), which defines communities on a global scope, considering the graph as a whole. This is reasonable in many cases, but it is also somewhat reductive; it ignores the specificity of the singleton community as a separate entity, with its own dynamic and characteristics. On the other hand, embedding algorithms aim at preserving the proximity of nodes. As a result, using them in the first

step of the described procedure will help preserve the local properties of communities, resulting in a more nuanced approach to the community detection problem.

Similarly, it is expected that these ideas will also improve the quality of the results obtained by **Leiden** algorithm—because of additional refinement stage, it gives significantly better results compared to **Louvain** algorithm, but still, it is a greedy algorithm with all the inherent issues mentioned above.

Starting any of the two community detection algorithms from a properly generated initial partition seems to be a good idea but there are two problematic issues that we need to deal with: selection of embedding $\mathcal{E}$ and selection of clustering algorithm. There are plenty of different embedding methods to choose from [8, 18, 14, 24], that measure the proximity between nodes in different manners, which makes the selection of the algorithm a demanding task, often requiring a domain expert knowledge or time-consuming experiments. One of the goals of this work is to look at various embedding algorithms and test their behaviour in this particular task in order to find the best solution to create guidance for future users. We use following embedding algorithms: **Locally Linear Embedding (LLE)** [39], **Laplacian Eigenmaps (LE)** [4], **deepWalk** [34], **node2vec** [15], **LINE** (**L**arge-scale **I**nformation **N**etwork **E**mbedding) [43], **SDNE** (**S**tructural **D**eep **N**etwork **E**mbedding) [48], **GraRep** [9] and **HOPE** (**H**igh-**O**rder **P**roximity preserved **E**mbedding) [31]. Selected algorithms vary in complexity level and the way of measuring the proximity, allowing us to cover many different scenarios.

For each embedding, we also want to compare the results with divergence scores obtained by the **CGE** (**C**omparing **G**raph **E**mbeddings) [22, 19]—unsupervised framework created to compare and asses different embeddings. The **CGE** framework computes for a given embedding two scores: global and local. The *CGE global score* verifies how well the embedding captures the edge densities within and between communities of the graph, i.e. if an embedding is a good predictor of graph's global structure on an aggregate level. To complement it, the *CGE local score* verifies the quality of the embedding for predicting the presence of edges in the graph, i.e. if an embedding is a good local predictor of individual entries of graph's adjacency matrix. We believe that this framework might become a useful tool, significantly simplifying the selection process of a suitable embedding.

The biggest advantage of the embedded representation of graph's nodes is the fact that the data is not discrete anymore. Because of that, we might use a broader array of clustering algorithms [46, 49]. However, finding a proper one might be challenging.

There are plenty of well-known, efficient, and scalable algorithms; they might result in vastly different behaviour of the initial partitioning. Density-based algorithms, such as **DBSCAN** (**D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise) [11] or **HDBSCAN** (**H**ierarchical **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise) [29], will cluster only the points occupying the same densely connected regions. In contrast, the points in the sparsely inhabited areas will be considered noise and will not be assigned to any cluster. Thus, the initial partition **A** will contain only nodes, which are almost surely parts of the same communities, leaving the more ambiguous nodes for **Louvain** or **Leiden** algorithm. Another possibility is to use distribution-based clustering methods, such as **Gaussian Mixture Models** (**GMM**) [38], or traditional, centroid-based methods, such as k**-Means** [27]. The most significant advantage of these algorithms is that they will not leave nodes without initial assignment, further reducing the ambiguity of the task. Moreover, they allow users to fine-tune the number of clusters in the initial partition **A**, making it possible to adjust the impact of embedding step on the final partitioning **P**.

The experiment presented in this work aims to find a proper, generalized recipe for the proposed three-step procedure. It is necessary to test and understand how embeddings and clustering algorithms

behave and interact in the context of community detection problem—especially if we do not want the proposed approach to become merely a thought experiment. Iterating between all the possible combinations of embedding methods and clustering algorithms without any prior is time-consuming and cumbersome. It will render this method unusable in any practical context.

3 Experiment Design

The main body of the experiment was written in Julia 1.7.0 programming language with additional code and packages written in Python 3.7.10. The code for execution and analysis of the experiments is available on the GitHub repository, and so are Jupyter notebooks used to generate plots and tables *.

The experimental design was as follows: at the beginning, a comprehensive family of graphs with various properties was generated using the **ABCD** model [23, 21, 20]. This random graph model generates networks with community structure and power-law distribution for both degrees and community sizes. We used the following parameter sweep: the number of nodes $n = 1000$, tail exponents of the power-law distributions for community sizes $\beta \in \{1.1, 1.5, 1.9\}$ and degree distributions $\gamma \in \{2.1, 2.5, 2.9\}$, community sizes $c_{\min} = 0.005n$ and $c_{\max} = 0.2n$, the minimum degree $\delta \in \{1, 2, 5\}$, the maximum degree $D = \sqrt{n}$ and, finally, we set the mixing parameter $\xi \in \{0.15, 0.25, 0.35, 0.5, 0.65, 0.75, 0.85\}$ that controls the level of noise in the resulting graph.

In the experiments, we use artificially created networks for one important reason—such models are fully controllable and they cover a variety of different network topologies. Another advantage of synthetic models is so-called *ground-truth* planted in them. We can define ground-truth as a set of attributes assigned to nodes of graph G that influence how edges, and as a result, communities in G, are formed. With such knowledge, the quality of obtained partitioning $\mathbf{P}$ can be easily assessed with standard clustering similarity measures.

Unfortunately, there are two major issues with measuring the similarity of ground-truth and partitioning $\mathbf{P}$; firstly, real-world networks rarely have predefined ground-truth. As a result, performance of algorithm relative to ground-truth in synthetic model is not easily translatable to the general case. Processes that create the community structure are different, depending on the phenomenon that given network represents. We cannot ensure that algorithm able to match ground-truth in one case will perform as good in another scenario. Secondly, one might argue if using the ground-truth is a proper approach [33] in general. We cannot ensure, neither theoretically nor practically, that some observed discrete-valued node attributes used as labels are indeed, a good representation of an underlying mechanism responsible for creation of communities.

Considering these facts, we evaluated the performance of community detection algorithms using the score function, namely modularity. Our goal is to find an algorithm that *maximizes* the modularity function. In our opinion, this approach will result in the most unbiased comparison of different algorithms. However, in the case of synthetic **ABCD** graphs, we also measured the similarity (using the **A**djusted **M**utual **I**nformation Index (**AMI**) [47]) of obtained partitioning and the ground-truth in order to discuss the differences between both approaches.

To further test the performance of **EC** methods, experiments using real-world networks were also conducted. The used datasets were initially collected for **GEMSEC** (**G**raph **E**mbedding with **S**elf **C**lustering) paper [40]. They represent mutual like networks among verified Facebook pages (edge denotes a situation when two pages give like to each other) —data was collected based on

*`https://github.com/bartoszpankratz/ECCD`

Facebook official page categories. Types of sites included TV shows, politicians, athletes, and artists among others. Table 1 summarizes the datasets, they vary in size, density and clustering strength. It is worth mentioning that dataset contains only the basic structure of the networks, without any additional metadata or ground-truth. As we mentioned previously, our goal is to find an algorithm that maximizes the modularity.

Table 1: Description of the empirical social networks used in the experiment, data is taken from [40].

Dataset	$\|V\|$	Density	Clustering Coefficient
Artists	50,515	0.0006	0.1140
Athletes	13,866	0.0009	0.1292
Celebrities	11,565	0.0010	0.1666
Companies	14,113	0.0005	0.1532
Government	7,057	0.0036	0.2238
Media	27,917	0.0005	0.1140
Politicians	5,908	0.0024	0.3011
TV Shows	3,892	0.0023	0.5906

Louvain, **Leiden**, and **ECG** algorithms were each run 50 times for every given graph. As a result, we were able to compute the average modularity and volatility of each algorithm. These statistics were later used as a baseline for the comparison with **EC** methods Then, every graph was embedded using the following algorithms from the Python *OpenNE* package [†]: **LLE**, **LE**, **deepWalk**, **node2vec**, **LINE**, **SDNE**, **GraRep** and **HOPE**. For each of the selected algorithms, the dimensions $d \in \{8, 16, 32, 64, 128, 256\}$ were tested for **ABCD** artificial graphs. For real networks, the value of d was extended to the range $\{8, 16, 32, 64, 128, 256, 512, 1024\}$.

For **deepWalk** and **node2vec**, we use the following fixed values of parameters: number of walks per node $r = 10$, length of the random walk $\ell = 100$ and context window size $k = 10$. Additionally, for **node2vec**, we tested values of parameters p and q in $[0.25, 0.50, 1, 2, 4]$. For **LINE**, we set the number of negative edges drawn from the noise distribution K to 5. In the case of the **SDNE** algorithm, we tested values of the reconstruction weight of the non-zero elements $\beta \in [2, 3, 4, 5, 6, 7, 8, 9, 10]$ and sizes of the hidden layer in $h \in [128, 256, 512]$. For **GraRep**, the order proximity k was in the range $[1, 2, 4, 8]$. Finally, the **HOPE** algorithm was run for all four proximity measures: *Katz index*, *Personalized PageRank*, *Common neighbours* and *Adamic-Adar*.

To find the most suitable clustering algorithm we tested the following three methods for every embedding $\mathcal{E}$: **k-Means**, **HDBSCAN** and **GMM**. For **HDBSCAN**, we fix the minimum cluster size to 2 and test the minimum sample size, the parameter measuring how conservative the clustering will be, ranging from 1 to 10 with step 1. The larger the value of minimum sample size is, the more nodes will be declared as noise, and only really similar ones will be merged into the clusters. For k-**Means** and **GMM**, a number of clusters k depends on the experiment. For artificial networks, we test k from 500 to 32 with step 2. For Facebook graphs, the number of clusters was set to $k = |V|/\ell$, where ℓ, the expected volume of each cluster $\mathcal{A}_i$, changes from 2 to 2048 with step 2, but cannot exceed $0.2 \times |V|$. Finally, every partition $\mathbf{A}$ was used as the initial partitioning for both **Leiden** and

[†] `https://github.com/thunlp/OpenNE`

Louvain algorithms. To achieve comparable results, both methods were run 50 times on every **A**.

4 Results

In this section, we discuss the results of the aforementioned experiments. Subsection 4.1 focuses on general results. We compare the performance of baseline algorithms (**Louvain**, **Leiden** and **ECG**) with **EC** extensions of **Louvain** and **Leiden** on the artificial networks generated by the synthetic **ABCD** model. It allow us to test a vast array of possible networks, with different topologies. In comparisons we use the *best performing* **EC** algorithm (if not otherwise stated), namely, a combination of embedding algorithm and clustering algorithm that results in the largest modularity increase, compared to the respective baseline—**Louvain** or **Leiden** algorithm. In the latter part of the section, we discuss the results in relation to ground-truth communities, namely, we compare the obtained partitions with the ground-truth for each graph, with **AMI** index. In Subsection 4.2, we discuss the impact of different embedding algorithms on the performance of **EC** method. Our goal is to find which embedding algorithms are the most suitable for the proposed procedure. Subsection 4.3 discusses how clustering algorithm impact the quality of the results. Finally, in Subsection 4.4 results of the experiments on real-world networks are presented.

4.1 Results on ABCD Networks

Table 2 presents the results for one representative set of parameters: $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$ and different values of ξ. The results obtained by **Louvain** after using a proposed initialization procedure are improved, which is not always the case for the **ECG** algorithm, which seems very sensitive to the graph's parametrization. Both **EC-Louvain** and **EC-Leiden** can improve over the vanilla **Louvain** or in the worst case scenario, return the same value of modularity. In some rare cases, **EC-Louvain** can achieve performance similar to **Leiden**, but in most cases, it gives small to mediocre improvement. What is most interesting is that adding the initial partitioning **A** to **Leiden** significantly improves its quality and reduces volatility.

Table 2: Modularity increase with respect to baseline (plain **Louvain**) for **ABCD** graphs with different values of ξ and fixed $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$. Column *Baseline Louvain* shows the average modularity index (modularity $\times$ 100) with standard deviation. Other columns present the modularity change relative to **Louvain** (in percentage). For **EC** methods, the best performing combination of embedding and clustering algorithms was chosen.

		Relative Change (in %)			
ξ	Baseline Louvain	ECG	Leiden	EC-Louvain	EC-Leiden
0.25	68.04 ± 0.05	-0.35 ± 0.01	0.05 ± 0.01	0.03 ± 0.03	0.06 ± 0.00
0.35	58.13 ± 0.50	0.05 ± 0.02	0.50 ± 0.04	0.25 ± 0.24	0.52 ± 0.00
0.5	45.26 ± 0.85	2.00 ± 0.12	3.52 ± 0.29	1.32 ± 0.70	4.24 ± 0.02
0.75	30.53 ± 0.36	-6.50 ± 0.28	6.40 ± 0.29	3.20 ± 0.45	10.14 ± 0.21

The impact of ξ, a *noise parameter* that controls the expected fraction of edges between

communities, on the performance of the algorithms is visible in Table 2. When ξ is small, we have a graph with strongly separated communities. When ξ is large, then communities are more blurred and mixed together. We see that the advantage of using the augmented methods is diminishing with a decreasing ξ, which seems natural—for small ξs task is easy enough for the baseline algorithms, so there is no need to augment them in any way. Similarly, when a change of the other structural parameters of **ABCD** model distorts the community structure of the graph, then the advantage of using the augmented methods is more visible as presented in Table 3.

Table 3: Modularity increase with respect to baseline (plain **Louvain**) for synthetic **ABCD** graphs with different parametrizations. Column *Baseline Louvain* shows the average modularity index (modularity $\times$ 100) with standard deviation. Other columns present the modularity change relative to **Louvain** (in percentage). For **EC** methods, the best performing combination of embedding and clustering algorithms was chosen.

(a) Different values of δ and fixed $\xi = 0.5$, $\beta = 1.5$ and $\gamma = 2.5$.

		Relative Change (in %)			
δ	Baseline Louvain	ECG	Leiden	EC-Louvain	EC-Leiden
1	91.50 ± 0.09	-0.79 ± 0.09	0.13 ± 0.08	0.01 ± 0.10	0.33 ± 0.03
2	56.47 ± 0.28	-8.96 ± 0.23	1.81 ± 0.29	0.19 ± 0.22	3.23 ± 0.15
5	45.26 ± 0.85	2.00 ± 0.12	3.52 ± 0.29	1.32 ± 0.70	4.24 ± 0.02

(b) Different values of β and fixed $\xi = 0.5$, $\gamma = 2.5$ and $\delta = 5$.

		Relative Change (in %)			
β	Baseline Louvain	ECG	Leiden	EC-Louvain	EC-Leiden
1.1	41.13 ± 1.16	5.29 ± 0.42	7.06 ± 1.19	2.05 ± 1.11	11.09 ± 0.04
1.5	45.26 ± 0.85	2.00 ± 0.12	3.52 ± 0.29	1.32 ± 0.70	4.24 ± 0.02
1.9	43.22 ± 1.00	2.12 ± 0.21	3.22 ± 0.54	1.20 0.89	4.91 ± 0.02

(c) Different values of γ and fixed $\xi = 0.5$, $\beta = 1.5$ and $\delta = 5$.

		Relative Change (in %)			
γ	Baseline Louvain	ECG	Leiden	EC-Louvain	EC-Leiden
2.1	44.87 ± 1.05	2.13 ± 0.14	2.93 ± 0.31	1.00 ± 0.77	(3.40 ± 0.01
2.5	45.26 ± 0.85	2.00 ± 0.12	3.52 ± 0.29	1.32 ± 0.70	4.24 ± 0.02
2.9	41.77 ± 0.73	1.95 ± 0.46	5.88 ± 0.66	0.84 ± 1.09	9.83 ± 0.07

When minimum degree is set to $\delta = 1$, the difficulty is significantly reduced because the substantial portion of nodes have only one neighbour and inevitably belong to the same community. But even then, **EC** methods improve over baseline algorithms. When δ grows, the problem becomes more

demanding and gains from using the augmented methods are more visible. But **EC-Leiden** has one interesting property: relative to the plain **Leiden**, the improvement of modularity gained by **EC-Leiden** decreases when δ grows. For $\delta = 1$, the gain from using **EC-Leiden** is about 2.7 times greater than the gain from the usage of regular **Leiden**. When $\delta = 5$, it is only 1.2 times. This shows an interesting property of the proposed framework. With a proper embedding selected, which learns non-trivial relationships in a given graph, it is possible to deal with even the most ambiguous nodes and refine the final partition far better than other methods.

β is a parameter of the power-law distribution used to generate the sequence of cluster sizes of every graph G. With a small β, e.g. equal to 1.1, we get a smaller number of communities with more evenly distributed nodes. When β grows, the number of communities rises, but they become smaller. When we look at the results, we can see that gains from using algorithms other than **Louvain** are larger when β is small. It is directly caused by the well-known phenomena of resolution limit [13] that can be summarized as follows: optimizing modularity function in large networks cannot find small communities, even if they are well defined. Sadly, even if the proposed solution can improve the results, it is still prone to this problem.

Finally, the impact of γ on the results might be explained somewhat similarly to the role of δ. γ is a parameter of the power-law distribution used to generate the degree distribution. When γ is small, degrees are much more evenly distributed in G. When γ is large, the vast majority of nodes have a relatively low degree (usually close or equal to δ) with only a small fraction of hubs . In this particular scenario, when using the plain **Louvain**, there is a serious risk that connectors (nodes with the majority of external edges, connecting different communities) will be wrongly assigned resulting in a poorly connected or even disconnected community. This problem was one of the main reasons for introduction of **Leiden** algorithm [44] and from Table 3c, we see that **Leiden** really improves over **Louvain**. However, for noisy graphs, the backtracking and refinement offered by **Leiden** is not enough; **EC-Leiden** further improves modularity by initially using the embedding and clustering and avoiding greedy enumeration over all neighbours, that is the source of problem.

Now, let us discuss how well obtained results match with the ground-truth of the **ABCD** model. Table 4 presents the results for one representative set of parameters: $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$ and different values of ξ. Similarly to the modularity comparison, **EC-Louvain** and **EC-Leiden** improve the plain versions of algorithms. Moreover, we see that the scale of the improvement depends on ξ – for small values of ξ the gain is nearly negligible whereas for noisy graphs, with large values of ξ, **EC** methods are far better than the respective baseline. However, interestingly, **ECG** outperforms **EC** methods in all cases and typically the difference between algorithms is quite substantial. Considering that **EC** methods are substantially better in modularity optimization, it might seem counteractive. However, there is a good reason why it happens.

ABCD is a stochastic model. Firstly, nodes are assigned into communities randomly, based on the degree distribution and the distribution of community sizes. Then, nodes are paired into edges also randomly. As a result, edge density is not homogeneous, neither between communities nor inside each community. As proven in [21], it has two important implications: a) even purely random graph exhibits some community structure and b) some subsets of each planted ground-truth community could be denser than the density of the community; in extreme cases they could be considered as a separated *community*. Other synthetic benchmark models and real-world networks possibly exhibit the same behaviour. As a result, the ground-truth might not coincide with a partition with the highest possible edge density within communities, thus modularity maximization is not equivalent to ground-truth retrieval task.

Table 4: Adjusted Mutual Information (**AMI**) increase with respect to baseline (plain **Louvain**) for **ABCD** graphs with different values of ξ and fixed $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$. Column *Baseline Louvain* shows the average AMI index (AMI $\times$ 100) with standard deviation. Other columns present the AMI change relative to **Louvain** (in percentage). For **EC** methods, the best-performing combination of embedding and clustering algorithms was chosen.

		Relative Change (in %)			
ξ	Baseline Louvain	ECG	Leiden	EC-Louvain	EC-Leiden
0.25	95.13 ± 0.48	4.87 ± 0.05	(0.40 ± 0.26	0.16 ± 0.40	0.73 ± 0.27
0.35	95.24 ± 1.26	4.99 ± 0.04	0.99 ± 0.21	0.25 ± 0.83	1.28 ± 0.24
0.5	79.86 ± 4.09	17.29 ± 0.67	9.84 ± 1.98	3.97 ± 3.62	14.32 ± 0.64
0.75	12.50 ± 0.93	41.53 ± 0.84	1.06 ± 0.96	5.75 ± 0.88	23.03 ± 0.86

This argument explains why **EC** methods, despite being better in maximizing modularity, perform worse relative to the ground-truth. They can identify densely interconnected sub-communities within larger communities and separate them. Answering the question of why **ECG** performs so well in the ground-truth retrieval on **ABCD** graph is far less obvious; it would require its own separate research focusing on the design of this algorithm, which is not in the scope of this research. Finally, it is worth mentioning that **EC** methods could be used with different score functions; there are plenty of possibilities [26, 50]. Some of them perform better, relative to the ground-truth, and it is worth experimenting with them in future research.

4.2 Performance of Embedding Algorithms

Figure 2 shows the relation between modularity and **CGE** scores obtained by the unsupervised framework for comparing graph embeddings for a single sweep of parameters ($\xi = 0.5$, $\beta = 1.5$, $\gamma = 2.5$ and $\delta = 5$). The framework assigns two scores, local and global, to each embedding that measure the quality of an evaluated embedding for tasks that require good representation of local and, respectively, global properties of the network. Embeddings with a lower score better describe the respective properties of the network. Both, local and global points on the plot represent the best performing embedding for different combinations of embedding algorithms and dimensionality. We could see that a correlation exists between the quality of the embedding and obtained score—stronger for **EC-Leiden**, weaker for **EC-Louvain**. Figure 2 also shows that in most cases, when we decide to use **EC** method, even if we do not select the best possible embedding, **EC** still improves the results over the baseline algorithm. In such a scenario, the gain will not be as large as in the best case. However, initializing modularity optimization algorithm with an initial partitioning **A** improves the results. What is most important, this plot confirms our presumptions about the dimensionality of embedding—it should be rather small, around 32.

But still, even if **CGE** scores help with detecting the best performing embeddings, running the procedure multiple times will be too tedious. Thus, it is necessary to dig deeper into the data and find emerging patterns that will further improve the user-friendliness of **EC** scheme. Figure 3 shows the results for the best embedding algorithms for all tested values of ξ, β and γ.

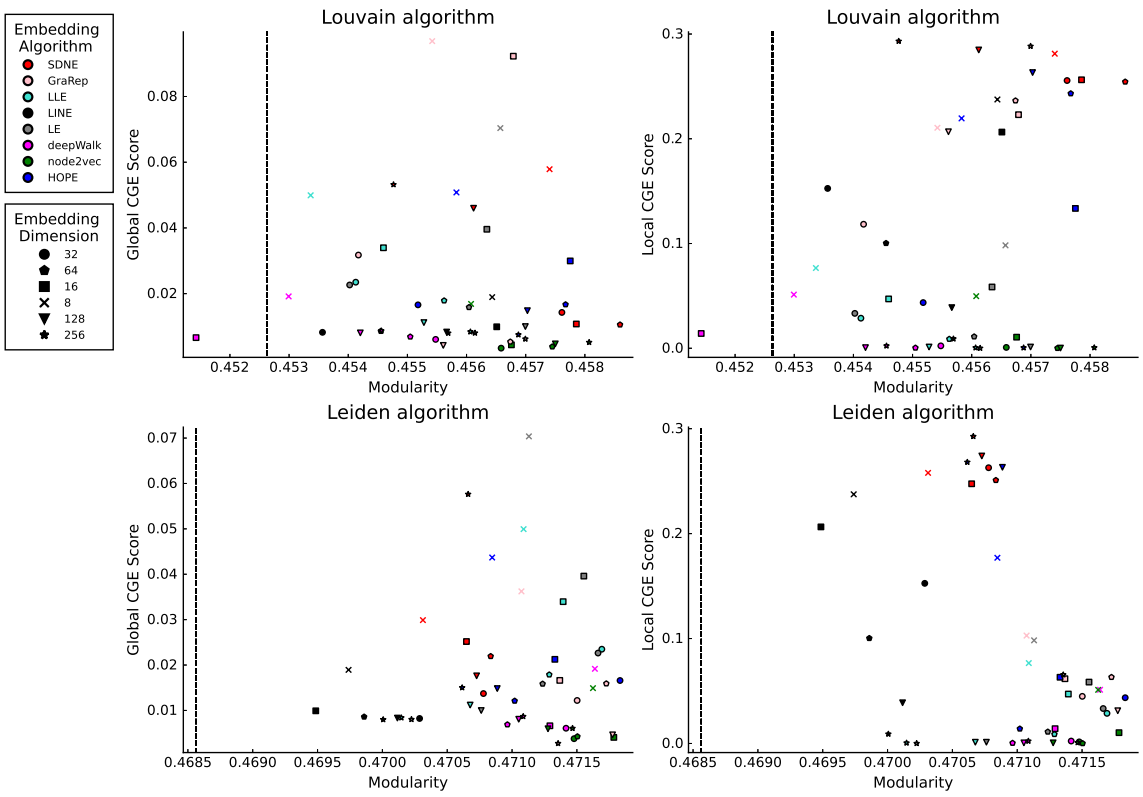
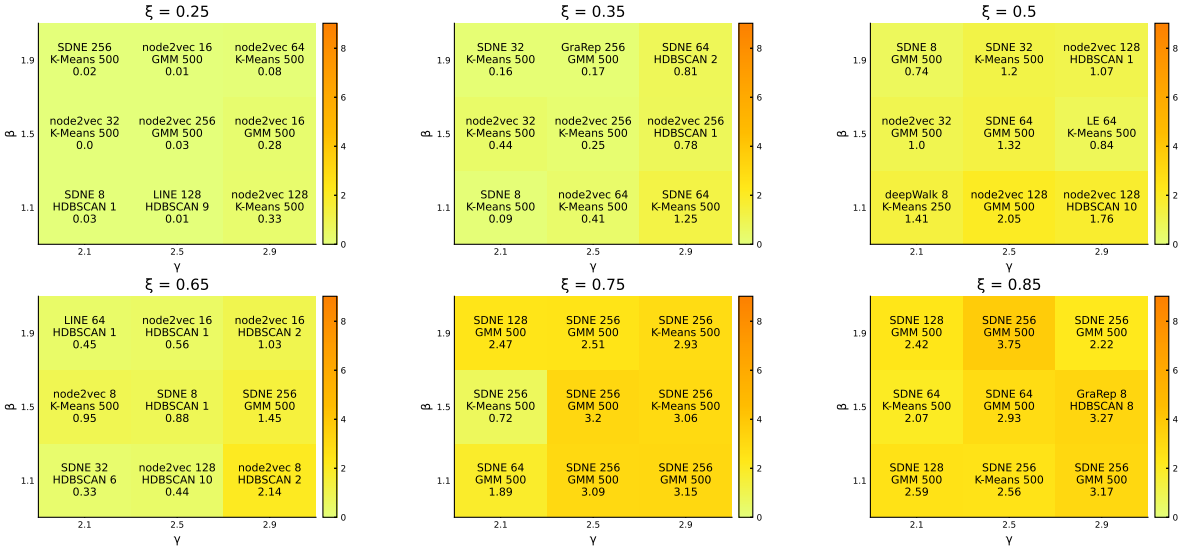


Figure 2: Modularity (dashed line represents modularity of the baseline algorithm) and Global/Local CGE scores (the lower score, the better) for different embeddings in **ABCD** graph with parameters: $\xi = 0.5$, $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$.

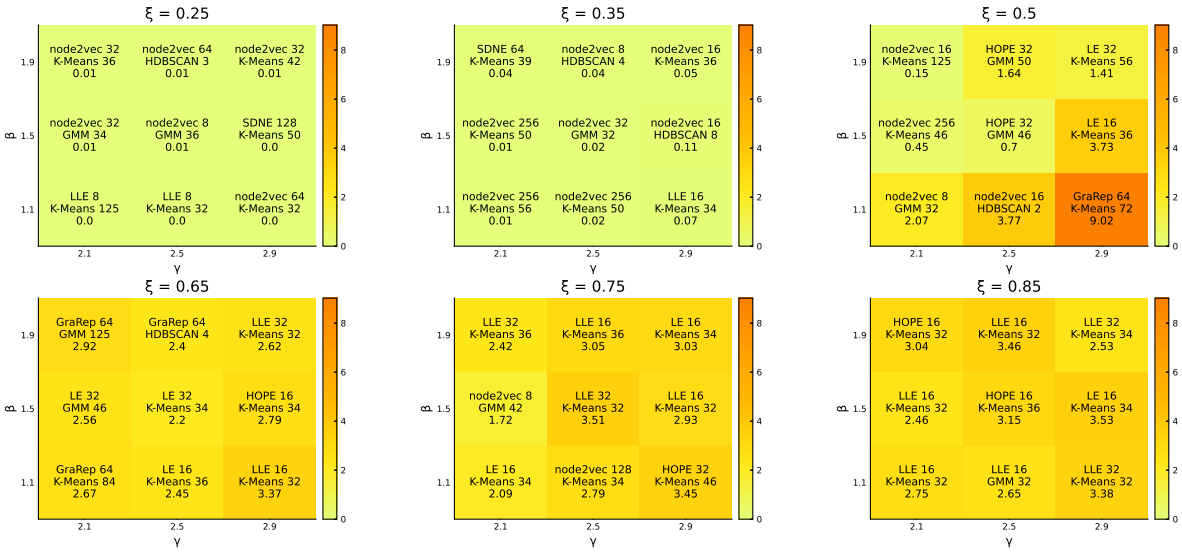
Figure 3 confirms the previously mentioned findings. The advantage of using **EC** methods is clearly growing with ξ and γ, while growing β negatively impacts the results. We also see a relation between the parametrization of the graph, embeddings and their dimensionality. For graphs with strongly separated communities, **node2vec** is usually the best performing one, although it offers a relatively small gain in modularity value. When ξ rises, we see higher gains, but also the best embeddings change gradually to simpler methods. In the case of **EC-Louvain** the best performing algorithm shifts from **node2vec** to **SDNE** and, in the case of **EC-Leiden**—from **node2vec** to **LLE** or **LE**. We will discuss embeddings later—firstly, let us focus on their dimensionality.

With some exceptions for small ξ we see that the best performing embeddings have rather small dimensionality. The only exception is **EC-Louvain** for $\xi \geq 0.75$, where the best results are given by a combination of the **SDNE** embedding and high dimensionality (usually 256). For any other embedding, we should generally choose a relatively small value of d, usually around 32. Our goal is to preserve the most important information about the node's role in G; thus, we choose moderately small d, which will filter the unnecessary information while not being too reductive.

As we expect, for a small ξ, gains are minimal, almost insignificant. But when graphs became noisier, some interesting patterns emerge. For both baselines, **node2vec** is the best performing algorithm for small ξ, followed by **HOPE** and **GraRep** for the larger values of ξ and **SDNE** for **EC-Louvain** and **LE** or **LLE** for **EC-Leiden** when we reach the highest values of ξ. **LINE** is



(a) EC-Louvain



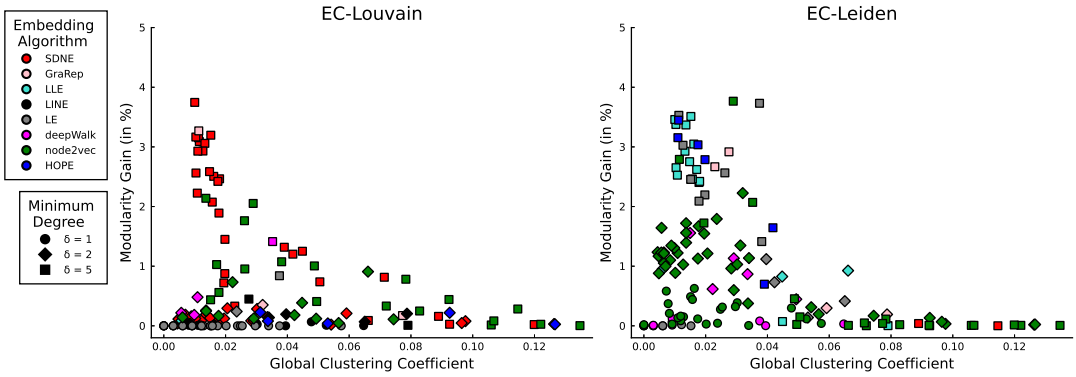
(b) EC-Leiden

Figure 3: Heatmap with relative modularity increase (in percentage) for various **ABCD** graphs. Each cell corresponds to **ABCD** graph with given parameters β, γ, ξ and fixed $\delta = 5$. Modularity increase is represented in a color scale mapping. Each cell also records embedding (together with its dimensionality) and clustering (together with its parametrization: *number of clusters* for k-**Means** and **GMM**, *minimum sample size* for **HDBSCAN**.)

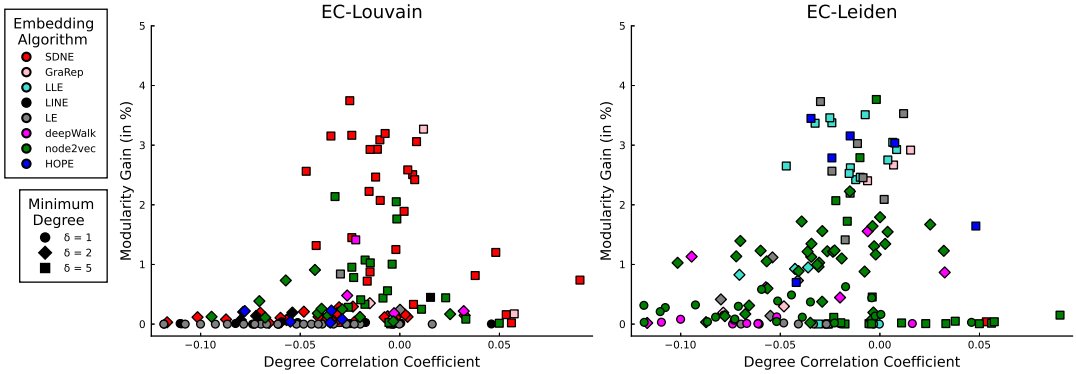
consecutively the worst performing algorithm, and **deepWalk** gives worse results than **node2vec** in all scenarios—we might think of it as a cruder version of **node2vec**—thus such performance is expected. What we see here is that we start with very sophisticated, random-walk based methods, then move to simpler algorithms, which still preserve the higher order proximities, and end up with the matrix factorization methods that preserve only the first order proximity. Even if we consider

the fact that **EC-Louvain** relies on the **SDNE**, the pattern still holds. **SDNE** is a variational autoencoder network that relies on the adjacency matrix to measure the proximity of the nodes. As such, **SDNE** can be understood as a form of a non-linear **LLE** and it is clear why it is used in conjunction with a more volatile **Louvain** algorithm, where **LLE** is sufficient enough for the stabler **Leiden**.

Experiments conducted for various properties of generated graphs give other interesting insights into the construction and behaviour of **EC** procedure. We focus on properties, that are the most easily computable and also have the strongest impact on the design, namely, selection of embedding algorithm. It is important, especially in the context of possible real-world applications, where it might be the only prior knowledge available. For a comprehensive comparison, please refer to the complementary Jupyter notebooks [‡].



(a) Global Clustering Coefficient



(b) Degree Correlation Coefficient

Figure 4: Modularity increase (in percentage) from various embedding-clustering combinations as a function of a) global clustering coefficient and b) correlation coefficient. **EC-Louvain** is plotted in the left column, **EC-Leiden** in the right.

Figure 4a shows the relation between the global clustering coefficient and the modularity gain given by **EC** method for all possible combinations of parameters β, γ, μ and δ. For a graph G, the global clustering coefficient is defined as the ratio of three times the number of triangles to the number of pairs of adjacent edges. It could be easily interpreted as the presence of the triadic

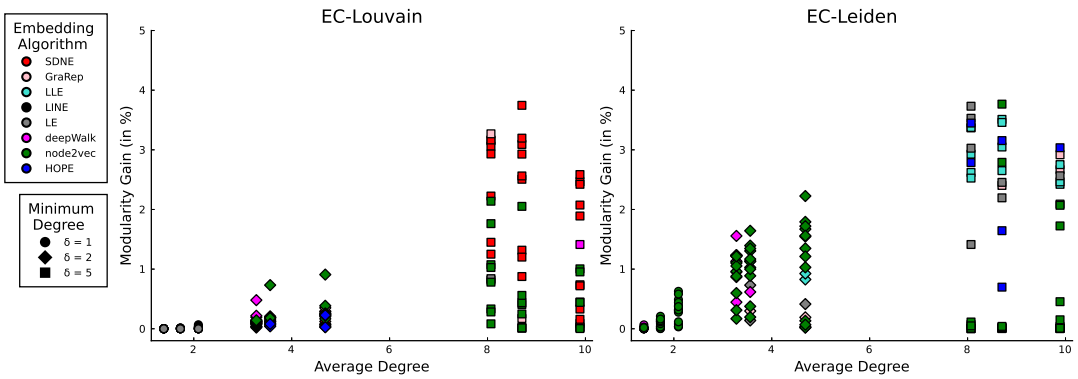
[‡] `https://github.com/bartoszpankratz/ECCD`

closures in the graph G – the probability that three nodes in a given random pair of adjacent edges will form a triangle (for further explanation see: [24], Chapter 1). Relationship between the global clustering coefficient and the modularity gain is negative, which is intuitive – higher value of the global clustering coefficient indicates a stronger tendency to cluster among nodes, thus an easier graph to work on for the baseline algorithms. However, the most interesting part of this plot is the relation between the global clustering coefficient, minimum degree δ and the selection of the best performing embedding algorithm. For both, **EC-Louvain** and **EC-Leiden** reductive embeddings (**SDNE** and **LE** or **LLE** respectively) are the best ones only in a very specific scenario - graphs with $\delta = 5$ and very low, close to 0, global clustering coefficient. In such a scenario, embedding in **EC** method is used to reduce the amount of unnecessary information held by the excess edges in the graph. In all the other cases, it is preferable to use the more sophisticated, random walk-based embedding, which will learn the role of each node in a broader context, that is unobtainable for simple, greedy optimization.

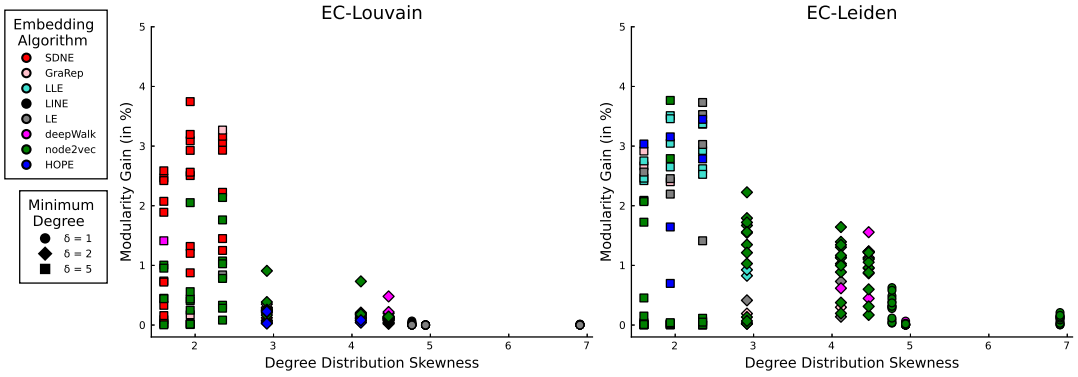
Similar conclusions came from the inspection of figure 4b. The degree correlation coefficient measures the strength of the overall assortativity of a graph G (see Chapter 4 in [24]). A positive value indicates an assortative network, where high degree nodes tend to be adjacent to other high degree nodes and low degree nodes link with other low degree nodes. A negative value is a mark of a disassortative network – one where high degree nodes tend to link with low degree nodes. Finally, a value close to zero represents a neutral network. For graphs with a strong degree correlation (either positive or negative) modularity gain is rather low, also the best performing embeddings are either **deepWalk** or **node2vec**. This is understandable, in the networks with a stronger degree correlation we could expect visible communities. Either in the form of nodes with similar degree clustering together (in assortative networks) or in the form of communities of high degree hubs, surrounded by low degree neighbours (in disassortative networks). As a result, such networks are easier tasks for the baseline community detection algorithms, thus **EC** methods are usable as a refinement tool. On the other hand, when a network displays a weak degree correlation, it resembles a random network, then **EC** methods are useful for filtering the data. This results in scenarios, where the gain from using **EC** methods is higher, but also the usage of more reductive embeddings is required.

Figure 5a shows the relation between a modularity gain and the average degree of a graph. With a growing average degree, gain from using **EC** method is growing substantially, also the shift in preferred embedding algorithm is clearly visible. This result confirms our previous intuition: for dense graphs (ones with a high average degree) we want to simplify the problem, removing possibly misleading information, thus we use more reductive, adjacency matrix based embedding methods. Otherwise, we want to use more complex embedding algorithms, that will learn non-trivial relations in the graph to refine the results of the baseline community detection algorithm. The same pattern is visible, when we look at figure 5b. When skewness of degree distribution is close to 0, then degrees in G are evenly distributed around the average degree, resulting in a graph that structurally somewhat resembles a random network. On the other hand, heavily positive skewed distribution indicates that communities in the network are built from large degree hub nodes, surrounded by low degree peripheral nodes. The latter situation naturally results in a more visible community structure, thus **EC** algorithms give lower modularity gain because they are used for improving and fine-tuning the results of the baseline algorithms.

Figure 6 shows the relation between modularity and **AMI** index for a single sweep of parameters ($\xi = 0.5$, $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$) for different embedding and clustering combinations. There is a strong, linear relationship between both measures. It means that the parametrization of



(a) Average Degree



(b) Skewness

Figure 5: Modularity increase (in percentage) from various embedding-clustering combinations as a function of a) average degree and b) skewness of degree distribution. **EC-Louvain** is plotted in the left column, **EC-Leiden** in the right.

EC method will perform similarly well on both tasks, despite the difference between modularity optimization and ground-truth retrieval described in Subsection 4.1. It means that **EC** are somewhat "measure-agnostic", namely we could use the same combination of embedding and clustering for different tasks or with different score functions and expect similarly good outcomes, which is a promising result.

4.3 Performance of Clustering Algorithms

Now, let us examine the role of clustering algorithms and their parameters in **EC** framework. Figure 3 shows the clustering algorithm with parametrization for the best **EC** method. In most cases, *k*-**Means** is the best-performing algorithm, followed by **GMM**. Cases when **HDBSCAN** outperforms the rest are rather rare. **HDBSCAN** was chosen mostly in the cases when the gain from **EC** method is almost negligible. Also, the selected values of *minimum samples* do not show any reasonable pattern. It may be explained by the fact that **HDBSCAN** is a density-based algorithm; by definition, **HDBSCAN** merges similar nodes into communities, leaving the rest as noise. It is quite likely that many of the merged nodes are neighbours in the graph that are also merged by the baseline algorithms, whereas the truly ambiguous nodes, challenging cases, will be left untouched.

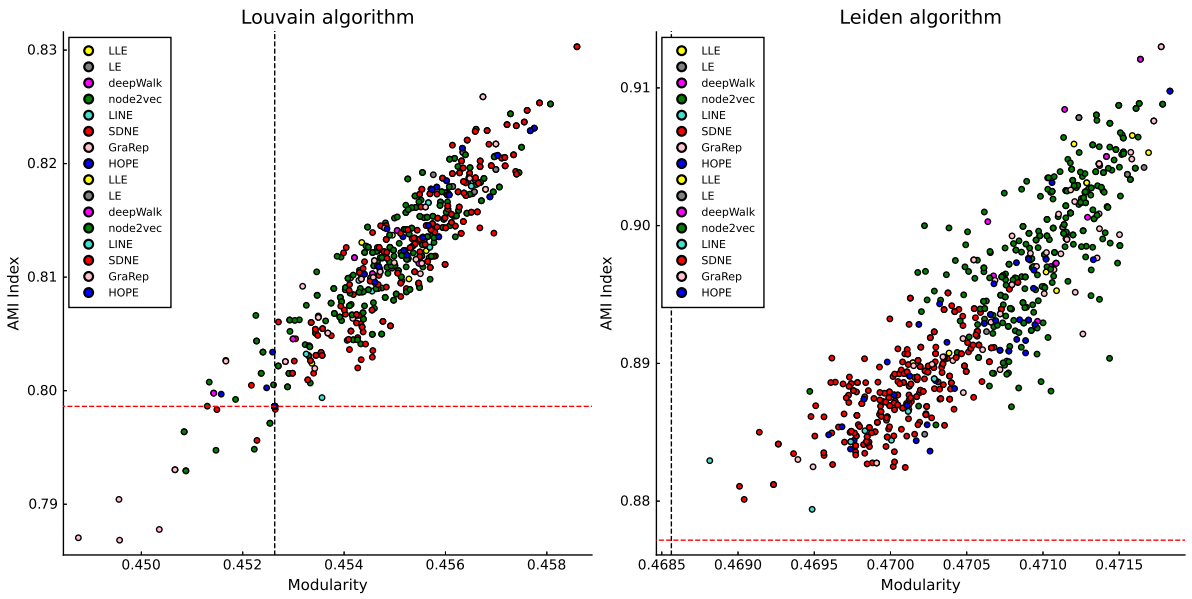


Figure 6: Modularity (dashed line represents modularity of the baseline algorithm) and AMI Index (dashed red line represents results for the baseline algorithm) for different embeddings in **ABCD** graph with parameters: $\xi = 0.5$, $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$.

As a result, **HDBSCAN** negates the impact of additional embedding and clustering steps.

As expected, performance of k-**Means** and **GMM** is similar. They serve the same role but the slight edge of k-**Means** might be explained by the fact that it does not introduce new bias in the form of the assumed probability distribution over given data (embedded nodes). In both clustering algorithms, the number of clusters k varies from 500 to 32; $k = 500$ gives us the most conservative scenario, merging only the two closest points into a cluster. With growing k, we significantly shrink and simplify the problem but we also rely gradually more and more on the embedding quality. Interestingly, there is a major difference between **EC-Louvain** and **EC-Leiden**—**EC-Louvain** almost exclusively achieves the best performance for $k = 500$, whereas **EC-Leiden** works best with the number of clusters k, roughly around 40.

4.4 Facebook Datasets

Table 5 summarizes results of experiments on Facebook Datasets. As in **ABCD** experiments, **EC** methods improve on the baseline methods and reduce volatility. Improvement does not depend on the size of a graph but on its properties. Gain from using the **EC** methods is larger when graphs are sparser and have a lower clustering coefficient.

As seen in Table 6, the best combinations of embedding and clustering algorithms follow a pattern resembling the results obtained on the artificial networks. What is worth mentioning, dimensionality of embedding does not depend on the size of the graph, but rather on the embedding algorithm and the complexity of the problem. For graphs with a higher clustering coefficient neural network-based algorithms (**SDNE** for **EC-Louvain** and **deepWalk** for **EC-Leiden**) with high dimensionality are preferred, whereas for simpler problems, the best performance is achieved for matrix factorization algorithms with small dimensionality. This observation is important, especially considering the fact

Table 5: Modularity increase with respect to baseline (plain **Louvain**) for Facebook networks. Column *Baseline Louvain* shows the average modularity index (modularity × 100) with standard deviation. Other columns present the modularity change relative to **Louvain** (in percentage). For **EC** methods, the best performing combination of embedding and clustering algorithms was chosen.

		Relative Change (in %)			
ξ	Baseline Louvain	ECG	Leiden	EC-Louvain	EC-Leiden
Artists	61.18 ± 0.52	0.79 ± 0.33	0.67 ± 0.34	1.03 ± 0.11	1.59 ± 0.01
Athletes	71.08 ± 0.33	-1.88 ± 0.14	1.37 ± 0.09	0.40 ± 0.14	1.59 ± 0.03
Celebrities	68.39 ± 0.11	-0.88 ± 0.07	0.66 ± 0.06	0.00 ± 0.13	0.87 ± 0.02
Companies	72.83 ± 0.27	-2.39 ± 0.10	1.58 ± 0.08	0.42 ± 0.19	1.93 ± 0.04
Government	72.74 ± 0.09	-1.57 ± 0.11	0.16 ± 0.06	0.04 ± 0.06	0.32 ± 0.01
Media	62.56 ± 0.21	-3.57 ± 0.13	1.85 ± 0.16	0.07 ± 0.14	2.11 ± 0.04
Politicians	86.82 ± 0.05	-1.40 ± 0.23	0.14 ± 0.04	0.05 ± 0.04	0.21 ± 0.00
TV Shows	87.20 ± 0.08	-1.38 ± 0.11	0.14 ± 0.01	0.04 ± 0.04	0.18 ± 0.01

that the bigger gain is achieved in the latter situation. It shows that **EC** methods can be used efficiently even for substantial networks, without becoming too computationally expensive.

For **EC-Louvain**, **k-Means** and **GMM**, with a large initial number of clusters are giving the best results. The value of ℓ, the expected volume of every subset $\mathcal{A}_i$, is close to 2 for almost all datasets, which follow the intuition presented in Section 4.3. The goal of clustering in **EC-Louvain** is to slightly truncate the initial graph and allow the **Louvain** algorithm to do the rest. **EC-Leiden** behave differently, ℓ is roughly equal to 64, similar to the expected volume of subsets in **ABCD** networks, despite differences in the size between **ABCD** networks and Facebook datasets.

5 Summary of the Procedure

Results of the numerical experiments show that both **EC** methods prove to be viable improvements over their baseline counterparts. Especially **EC-Leiden** turned out to improve the performance by a fair margin. Thus, we suggest using it, rather than **EC-Louvain** algorithm, which is less stable and gives worse overall results. For a graph $G = (V, E)$ we could summarize **EC-Leiden** steps as follows:

Step 1: Compute some basic descriptive statistics of G (density, clustering coefficient, degree correlation coefficient, etc.). For dense G with a low clustering coefficient and degree correlation coefficient around 0, embed each node of graph G into a low-dimensional (e.g. 32) vector using the **LE** or **LLE** embedding algorithm. Otherwise, use **node2vec** algorithm [§] with a similarly small dimensionality.

Step 2: Run **k-Means** clustering algorithm on the obtained latent representation $\mathcal{E}$ to get partition $\mathbf{A} = \{A_1, A_2, \ldots, A_k\}$. **EC-Leiden** gives the best results when k, size of partition

[§]In a case of **node2vec**, hyperparameters do not impact the results in a significant manner, thus we suggest using the parametrization proposed in [15] or a default one in the preferred implementation of **node2vec**

Table 6: Best performing **EC** methods for Facebook datasets. Columns *Embedding Algorithm* and *Dimensions* show the best embedding algorithm and its dimensions. Columns *Clustering Algorithm* and *Parameters* show the best clustering algorithm and its parametrization (*number of clusters* for *k*-**Means** and **GMM**, *minimum sample size* for **HDBSCAN**.

(a) EC-Louvain

Dataset	Embedding Algorithm	Dimensions	Clustering Algorithm	Parameters
Artists	node2vec	16	GMM	8420
Athletes	HOPE (ppr)	8	K-Means	6933
Celebrities	SDNE	64	GMM	5783
Companies	SDNE	1024	HDBSCAN	4
Government	LINE	1024	K-Means	3529
Media	LLE	256	K-Means	13959
Politicians	SDNE	64	HDBSCAN	1
TV Shows	SDNE	1024	HDBSCAN	2

(b) EC-Leiden

Dataset	Embedding Algorithm	Dimensions	Clustering Algorithm	Parameters
Artists	LLE	128	GMM	395
Athletes	LLE	16	K-Means	55
Celebrities	LLE	128	K-Means	181
Companies	node2vec	8	GMM	442
Government	LE	32	K-Means	111
Media	node2vec	32	GMM	931
Politicians	LE	32	K-Means	93
TV Shows	deepWalk	1024	K-Means	217

A, is roughly equal to size of the final partition **P**. However, the results are still reasonable when k is just high enough - we suggest setting the value of $k = |V|/64$, thus assuming that each community will contain roughly 64 nodes .

Step 3: Run **Leiden** algorithm on graph G using partition **A** as a starting point. The result of this procedure, partition **P**, is the outcome of our algorithm.

Let us discuss The difference between **EC-Louvain** and **EC-Leiden** and explain why **EC-Leiden** is a more suitable procedure.

Louvain merges two nodes if such a move maximizes modularity locally, without any broader context. It is especially problematic in the early stages, when the structure is still unclear because the

merge of nodes belonging to different communities might (and most often does) result in the largest possible increase of the target function. The obvious drawback here is that **Louvain** algorithm cannot later separate the nodes if they are wrongly connected.

The initial partitioning **A** was designed to overcome this issue, guaranteeing the stability of the first step of the algorithm and reducing the impact of wrongly connected nodes. **EC-Louvain** aims to find a stable initial partitioning, that is, merge nodes that are certainly part of the same community into one cluster. However, if there is an error in initial partitioning, there is no chance that the algorithm will be able to correct it: it will rather exaggerate it. And we know that all embeddings have some inevitable representation error, thus it is better to use a more sophisticated algorithm (e.g. non-linear **SDNE** instead of linear **LLE**) with higher dimensionality. Also, the initial number of clusters k is rather high—again, to avoid any possible information loss only certain nodes are connected into communities in the initial step.

The problem of inherent greediness of **Louvain** algorithm prevails in later steps until the algorithm reaches the stage when the communities are large enough. As a result, the impact of the initial "good" partitioning is minimized. This problem might be fixed by repeating the embedding process after every iteration, up to the moment when the algorithm reaches its stable stage, but obviously, such a procedure would be unfeasible for large graphs as it is very time-consuming.

In the case of **Leiden** algorithm, the refinement stage is designed to solve the aforementioned issue of wrongly connected nodes. After every iteration, when communities are created in the same manner as in **Louvain**, they are split and recombined into new, better partitions, ensuring that all nodes are optimally assigned in the context of a given subgraph induced by a single community. The scope of such refinement is somewhat limited; it cannot backtrack more than a single iteration. As a result, in the early stages, when initialized with a singleton partition, it might still merge nodes that should not belong to the same community—and that will be irreversible. By initializing it with a fine-tuned initial partitioning **A**, we reduce ambiguity in the crucial first step by giving an algorithm some base structure to work with and allowing the refinement stage to show its importance.

The addition of the refinement stage in **Leiden** algorithm means that **Leiden** will not amplify the inherent error caused by embedding, but it might also be able to reduce it. So, in this case, we might start with a rather simple and reductive embedding and already quite an aggregated initial partitioning **A**. This split does not need to be overly complicated and accurate because it will be further refined by **Leiden**, and then sewn together into a final, optimal partition **P**. Thus, the best strategy is to use simpler methods than in **EC-Louvain**, e.g. matrix factorization methods (**LE** or **LLE**) instead of the neural network-based **SNDE**, with small dimensionality d. Also, the initial number of clusters k is smaller than in **EC-Louvain** case, usually close to the final number of communities ℓ. In this context, **EC-Leiden** is a method that handles a non-trivial, NP-hard optimization problem by dividing it into a set of far more manageable smaller independent sub-problems, solving them, and then aggregating them into the final solution.

Finally, let us focus on the last interesting pattern, which is the behaviour of the best performing embedding algorithms in relation to the community structure of a given graph. In the cases of both **EC** methods, when the community structure is strong, communities are visible and separated, then the best strategy is to use rather sophisticated algorithms with a proximity measure able to capture the structural role of each node, such as **node2vec**. When a graph is "noisy", with less pronounced communities and many edges between them, then the best solution is to rely on simpler adjacency matrix-based methods—**SDNE** for **EC-Louvain** and **LE** or **LLE** for **EC-Leiden**.

When the graph exhibits a strong community structure, baseline algorithms already give good

results and high modularity value. The reason why we want to use **EC** is to properly assign the small fraction of the ambiguous nodes, that join two or more different communities. Thus, we are using an algorithm that can learn complicated and deep relations between nodes but offers a rather minuscule gain of modularity. For graphs with a high level of "noise" we want to achieve a somewhat opposite goal. Nodes usually have more neighbours belonging to many different communities. In this case, we are interested in filtering the excessive information and leave only the crucial one. Our goal is to embed nodes in a way that will ensure that closely related (adjacent or sharing common neighbours) nodes will not be embedded far apart without accounting for the more distant relationships and structural role of a node in the graph. In such case, **EC** method with simple, reductive embedding will result in significant modularity gain compared to the baseline.

6 Discussion

The most important advantage of the proposed **EC** method is its flexibility. **EC** algorithms use two different measures to achieve the final partitioning. Firstly, they use *local* distances between the embedded nodes and then modularity which measures the strength of communities **globally**. Measuring community structure in two ways improves the results in two ways, either by refining the results via learning more complex relations between nodes or by simplifying the structure of a graph, depending on selected embedding algorithms.

On the other hand, multilevel methods rely on aggregating the results of different runs of modularity-based algorithms in the form of a weighted graph that is later refined into the final partition **P**. Usually, a weighted graph is created using the low-level coarsening of the input graph G. They indeed reduce the volatility of the baseline algorithm, but they are not able to adjust their way of acting depending on the properties of the graph.

As a result, multilevel methods are unable to overcome inherent issues of the modularity optimization approach. Especially in the early steps, merging nodes from different communities will result in significant modularity gain, thus we might expect that greedy modularity optimization will often assign nodes to wrong communities. The issue is especially visible in two cases: when nodes have neighbours in many different communities (that is, when the graph is "noisy") and when the graph contains a significant amount of low-degree nodes (see Table 2 and Table 3a) that might have neighbours in different communities.

When the multilevel method creates weights, it looks at how often a node v_i is placed in the same community as its neighbour v_j. The weight w_{ij} between nodes v_i and v_j is large when v_i and v_j are often paired, and small when they are rarely merged by the algorithm. In the aforementioned two cases, we might expect that ambiguous nodes will have a close to uniform distribution of weights, but only in cases when the number of initial partitioning k is large enough. When k is too small, the multilevel algorithm amplifies volatility, resulting in ill-defined weights that perform worse than modularity optimization on the unweighted graph. On the other hand, creating the initial partitioning **A** on the embedded representation of G will always filter data and reduce volatility.

However, in its current form, **EC** methods have one drawback—their computational complexity might be overwhelming for practitioners who want to use such algorithms on a daily basis. However, in many cases the computational power is currently not a major constraint when doing graph analysis, especially when cloud computing resources are available. Moreover, there are some specific scenarios when we suggest using **EC** methods, even if computation resources are scarce.

First and the most obvious situation is finding communities in dense graphs with evenly distributed

degrees—graphs that we consider "noisy". The sheer amount of edges and lack of visible hub nodes in such networks make them a hard task for traditional community detection algorithms. Their results exhibit significant volatility, forcing users to run given algorithm multiple times to find the acceptable partition. Embedding step in **EC** methods reduces the complexity of a problem, allowing the baseline algorithm to find a better, far more robust and repetitive, partitioning of a given network.

Secondly, there are situations when properly aligning the most ambiguous nodes is crucial. For example, nodes that connect many different communities play a key role in propagation of disinformation and fake news in social media. Thus, placing such nodes in a proper community (e.g. together with other misinformation spreading accounts) is important to prevent the further spread of potentially dangerous and misleading news. In this scenario, even if the difference in modularity between a baseline algorithm and **EC** method might seem negligible, the impact of the improved solution is notable.

Finally, **EC** methods can be used in a situation when one already has a proper embedded representation prepared for another task, e.g. training a neural network. In this situation, the computational complexity of **EC** method, caused by the necessity of computing an embedding, is negligible. As a result, by using **EC** method one might improve the community detection results without significant additional cost.

7 Conclusions

The **Embedding-Clustering** (**EC**) scheme introduced in this paper improves the results of popular community detection algorithms. Using the initial partitioning C obtained by clustering nodes in graph embeddings improves the results of the popular community detection algorithms. In the case of **Louvain** the impact is rather small, almost negligible, but the initial partitioning of **Leiden** significantly improves its performance and reduces the volatility. Moreover, we show which classes of embeddings and clustering algorithms are the most suitable for this particular task.

Acknowledgements

Hardware used for the computations was provided by the SOSCIP consortium[¶]. Launched in 2012, the SOSCIP consortium is a collaboration between Ontario's research-intensive post-secondary institutions and small- and medium-sized enterprises (SMEs) across the province. Working together with the partners, SOSCIP is driving the uptake of AI and data science solutions and enabling the development of a knowledge-based and innovative economy in Ontario by supporting technical skill development and delivering high-quality outcomes. SOSCIP supports industrial-academic collaborative research projects through partnership-building services and access to leading-edge advanced computing platforms, fuelling innovation across every sector of Ontario's economy.

References

[1] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, STOC

[¶] `https://www.soscip.org/`

'00, page 171–180, New York, NY, USA, 2000. Association for Computing Machinery. doi:10.1145/335305.335326.

[2] Samin Aref, Hriday Chheda, and Mahdi Mostajabdaveh. The bayan algorithm: Detecting communities in networks through exact and approximate optimization of modularity, 2023. arXiv:2209.04562.

[3] Thomas Bartz-Beielstein and Martin Zaefferer. Model-based methods for continuous and discrete global optimization. *Applied Soft Computing*, 55:154–167, 2017. doi:10.1016/j.asoc.2017.01.039.

[4] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 585–591, Cambridge, MA, USA, 2001. MIT Press. doi:10.5555/2980539.2980616.

[5] Gema Bello Orgaz, Julio Hernandez-Castro, and David Camacho. Detecting discussion communities on vaccination in twitter. *Future Generation Computer Systems*, 66, 07 2016. doi:10.1016/j.future.2016.06.032.

[6] 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(10):10008, 2008. doi:10.1088/1742-5468/2008/10/P10008.

[7] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2), 2008. doi:10.1109/TKDE.2007.190689.

[8] Hongyun Cai, Vincent Zheng, and Kevin Chang. A comprehensive survey of graph embedding: Problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30, 09 2017. doi:10.1109/TKDE.2018.2807452.

[9] Shaosheng Cao, Wei Lu, and Qiongkai Xu. GraRep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 891–900, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2806416.2806512.

[10] William Deitrick and Wei Hu. Mutually enhancing community detection and sentiment analysis on twitter networks. *Journal of Data Analysis and Information Processing*, 01:19–29, 01 2013. doi:10.4236/jdaip.2013.13004.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231, Palo Alto, California, USA, 1996. AAAI Press. doi:10.5555/3001460.3001507.

[12] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, feb 2010. doi:10.1016/j.physrep.2009.11.002.

[13] Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, Dec 2006. doi:10.1073/pnas.0605965104.

[14] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, Jul 2018. doi:10.1016/j.knosys.2018.03.022.

[15] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2939672.2939754.

[16] Roger Guimerà and Luís Amaral. Functional cartography of complex metabolic networks. *Nature*, 23:22–231, 01 2005. doi:10.1038/nature03288.

[17] Leanne S. Haggerty, Pierre-Alain Jachiet, William P. Hanage, David A. Fitzpatrick, Philippe Lopez, Mary J. O'Connell, Davide Pisani, Mark Wilkinson, Eric Bapteste, and James O. McInerney. A Pluralistic Account of Homology: Adapting the Models to the Data. *Molecular Biology and Evolution*, 31(3):501–516, 11 2013. doi:10.1093/molbev/mst228.

[18] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40(3):52–74, 2017.

[19] Bogumił Kamiński, Łukasz Kraiński, Paweł Prałat, and François Théberge. A multi-purposed unsupervised framework for comparing embeddings of undirected and directed graphs. *Network Science*, page 1–24, 2022. doi:10.1017/nws.2022.27.

[20] Bogumił Kamiński, Tomasz Olczak, Bartosz Pankratz, Paweł Prałat, and François Théberge. Properties and performance of the ABCDe random graph model with community structure. *Big Data Research*, 30:100348, 2022. doi:10.1016/j.bdr.2022.100348.

[21] Bogumił Kamiński, Bartosz Pankratz, Paweł Prałat, and François Théberge. Modularity of the ABCD random graph model with community structure. *Journal of Complex Networks*, 10(6), 12 2022. doi:10.1093/comnet/cnac050.

[22] Bogumił Kamiński, Paweł Prałat, and François Théberge. An unsupervised framework for comparing graph embeddings. *Journal of Complex Networks*, 8(5):cnz043, 2020. doi:10.1093/comnet/cnz043.

[23] Bogumił Kamiński, Paweł Prałat, and François Théberge. Artificial benchmark for community detection (ABCD)—fast random graph model with community structure. *Network Science*, pages 1–26, 2021. doi:doi:10.1017/nws.2020.45.

[24] Bogumił Kamiński, Paweł Prałat, and François Théberge. *Mining Complex Networks*. Chapman and Hall/CRC, New York, 2021. doi:10.1201/9781003218869.

[25] Andrea Lancichinetti and Santo Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80(5), Nov 2009. doi:10.1103/physreve.80.056117.

[26] Jure Leskovec, Kevin J. Lang, and Michael Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 631–640, New York, NY, USA, 2010. Association for Computing Machinery. doi:10.1145/1772690.1772755.

[27] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982. doi:10.1109/TIT.1982.1056489.

[28] Arya D. McCarthy, Tongfei Chen, and Seth Ebner. An exact no free lunch theorem for community detection. In Hocine Cherifi, Sabrina Gaito, José Fernendo Mendes, Esteban Moro, and Luis Mateus Rocha, editors, *Complex Networks and Their Applications VIII*, pages 176–187, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-36687-2_15.

[29] Leland McInnes, John Healy, and Steve Astels. HDBSCAN: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017. doi:10.21105/joss.00205.

[30] Mark E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, May 2006. doi:10.1073/pnas.0601602103.

[31] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1105–1114, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2939672.2939751.

[32] Bartosz Pankratz, Bogumił Kamiński, and Paweł Prałat. Community detection supported by node embeddings (searching for a suitable method). In Hocine Cherifi, Rosario Nunzio Mantegna, Luis M. Rocha, Chantal Cherifi, and Salvatore Micciche, editors, *Complex Networks and Their Applications XI*, pages 221–232, Cham, 2023. Springer International Publishing. doi:10.1007/978-3-031-21131-7_17.

[33] Leto Peel, Daniel B. Larremore, and Aaron Clauset. The ground truth about metadata and community detection in networks. *Science Advances*, 3(5):e1602548, May 2017. doi:10.1126/sciadv.1602548.

[34] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk. *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, Aug 2014. doi:10.1145/2623330.2623732.

[35] Carlos A. R. Pinheiro. Community detection to identify fraud events in telecommunications networks. In *SAS SUGI Proceedings: Customer Intelligence*, 2012.

[36] Valérie Poulin and François Théberge. Ensemble clustering for graphs: comparisons and applications. *Applied Network Science*, 4(1), Jul 2019. doi:10.1007/s41109-019-0162-z.

[37] Tahereh Pourhabibi, Kok-Leong Ong, Booi H. Kam, and Yee Ling Boo. Fraud detection: A systematic literature review of graph-based anomaly detection approaches. *Decision Support Systems*, 133:113303, 2020. doi:10.1016/j.dss.2020.113303.

[38] Carl Edward Rasmussen. The infinite gaussian mixture model. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 554–560, Cambridge, MA, USA, 1999. MIT Press. doi:10.5555/3009657.3009736.

[39] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000. doi:10.1126/science.290.5500.2323.

[40] Benedek Rozemberczki, Ryan Davies, Rik Sarkar, and Charles Sutton. GEMSEC: Graph embedding with self clustering. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '19, page 65–72, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3341161.3342890.

[41] Didi Surian, Dat Quoc Nguyen, Georgina Kennedy, Mark Johnson, Enrico Coiera, and Adam G Dunn. Characterizing twitter discussions about HPV vaccines using topic modeling and community detection. *J Med Internet Res*, 18(8):e232, Aug 2016. doi:10.2196/jmir.6045.

[42] Aditya Tandon, Aiiad Albeshri, Vijey Thayananthan, Wadee Alhalabi, Filippo Radicchi, and Santo Fortunato. Community detection in networks using graph embeddings. *Phys. Rev. E*, 103:022316, Feb 2021. doi:10.1103/PhysRevE.103.022316.

[43] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale information network embedding. *Proceedings of the 24th International Conference on World Wide Web*, May 2015. doi:10.1145/2736277.2741093.

[44] Vincent Traag, Ludo Waltman, and Nees Jan van Eck. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, 9:5233, 03 2019. doi:10.1038/s41598-019-41695-z.

[45] Beethika Tripathi, Srinivasan Parthasarathy, Himanshu Sinha, Karthik Raman, and Balaraman Ravindran. Adapting community detection algorithms for disease module identification in heterogeneous biological networks. *Frontiers in Genetics*, 10:164, 2019. doi:10.3389/fgene.2019.00164.

[46] Sandro Vega-Pons and Jose Ruiz-Shulcloper. A survey of clustering ensemble algorithms. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(03):337–372, 2011. doi:10.1142/S0218001411008683.

[47] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(95):2837–2854, 2010. URL: http://jmlr.org/papers/v11/vinh10a.html.

[48] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1225–1234, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2939672.2939753.

[49] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16:645 – 678, 06 2005. doi:10.1109/TNN.2005.845141.

[50] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2350190.2350193.