Community Detection Supported by Node Embeddings
(Searching For a Suitable Method)

Bartosz Pankratz[1,2], Bogumił Kamiński[2], and Paweł Prałat[1]

[1] Department of Mathematics, Toronto Metropolitan University, Toronto, ON,
bartosz.pankratz@ryerson.ca
pralat@ryerson.ca
[2] Decision Analysis and Support Unit, SGH Warsaw School of Economics, Warsaw, Poland
bpankra@sgh.waw.pl
bkamins@sgh.waw.pl

Abstract. 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 tend to exhibit significant volatility. These issues might be solved by a proper initialization of such algorithms with some carefully chosen partition of nodes.
In this paper, we investigate the impact of such initialization applied to the two most commonly used community detection algorithms: **Louvain** and **Leiden**. We use a partition obtained by embedding the nodes of the graph into some high dimensional space of real numbers and then running a clustering algorithm on this latent representation. We show that this procedure significantly improves the results. Proper embedding filters unnecessary information while retaining the proximity of nodes belonging to the same community. As a result, clustering algorithms ran on these embeddings merge nodes only when they are similar with a high degree of certainty, resulting in a stable and effective initial partition.

Keywords: machine learning, community detection, complex networks, network embedding methods

1 Introduction

The main trait of most empirical complex networks is the fact that they tend to display a modular organization where one can easily separate sets of nodes (*subgraphs*) with considerably larger density of edges between nodes in such sets than between two different sets. This property is widely referred to as a community structure [8]. Finding such partitions is interesting not only from a theoretical perspective. Indeed, often communities that are extracted, or nodes inside them, exhibit different properties than the entire graph, so identifying them might give a meaningful insight into the data. However, in most cases such

underlying structure is unknown beforehand, thus we must use an unsupervised algorithm that is able to detect it. There are many existing solutions; the most common ones are built around a heuristic optimization of some carefully chosen score function.

Communities are somewhat elusive; without the full knowledge about the 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 [25]. This problem is widely discussed, see, for example, [19, 34, 21], and plenty of different score functions were proposed up to them. The modularity function [23] is possibly the most often used one.

Modularity measures the difference between the number of the edges within groups induced by a given partition $\mathcal{A}$ and the expected number of such edges given by an appropriately selected null-model, usually **Chung-Lu** random graph model [1]. For a graph $G = (V, E)$ and a given partition $\mathcal{A} = \{A_1, A_2, \ldots, A_\ell\}$, the modularity function is defined as follows:

$$q_G(\mathcal{A}) = \frac{1}{|E|} \sum_{A_i \in \mathcal{A}} \Big(e_G(A_i) - \mathbb{E}_{G' \sim \mathcal{G}(d)}[e_{G'}(A_i)] \Big), \tag{1}$$

where $|E|$ is the number of edges in G, $e_G(A_i) = |\{v_j v_k \in E : v_j, v_k \in A_i\}|$ is the number of edges in the subgraph of G induced by set A_i, and $\mathbb{E}_{G' \sim \mathcal{G}(d)}[e_{G'}(A_i)]$ is the corresponding expectation in the null-model.

However, optimizing modularity function is a NP-hard problem [5]; thus, basically all proposed solutions are heuristic in nature. One of the most popular, fastest, and best performing [18] ones is the **Louvain** algorithm [4]. Its core idea is simple yet effective, it is a two-step technique: it first moves each node to the community that provides the largest increase of the score function, ensuring that the score will be locally optimal; during the second step, it aggregates the communities into super-nodes. Then, both phases are repeated until there is no improvement of the score function. By default, the **Louvain** algorithm starts from a singleton partition in which each node belongs to its own community but it is possible to initialize the algorithm with a preexisting partitioning.

Despite the fact that **Louvain** is a great algorithm, it has some serious and known drawbacks. First, the obtained results are heavily stochastic, that is, each run of the algorithm on the same network may lead to the vastly different partitions. Moreover, it may create a weakly connected or even internally disconnected communities [32]. 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 that 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 which makes 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 by **Leiden** algorithm [32] 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) algorithm [27].

The latter idea is the center of this work, namely, we want to propose a method of community detection based on the modularity optimization with a spectral clustering initialization step. The procedure starts with an embedding of the nodes of the graph in the high dimensional space of real numbers, then the clustering algorithm is run on the obtained representation. The algorithm is fine-tuned to obtain many small clusters where only nodes that are very close in the latent space are merged together. As a result, we obtain a stable partition which is finally used to initialize the **Louvain** algorithm. In the same manner, such initial partition might also be used to improve other greedy optimization algorithms such as the **Leiden** one. In the experiments presented in this paper, we will test both of them but when describing the reasoning behind the proposed method we will use the **Louvain** algorithm as an example.

Our motivation is simple; we believe that 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. Having said that, relying only on the embedded representation is causing 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 that we propose in this paper is to combine both methods.

The goal of this paper is to test this premise. In order to do this, we perform an experiment aimed to answer the following four questions:(1) *How the proposed method performs compared to the other extensions of the* ***Louvain*** *algorithm (****Leiden*** *and* ***ECG****)?* (2) *How stable is the proposed method? How volatile are the results compared to the* ***Louvain*** *algorithm?* (3) *Is this method able to improve the* ***Leiden*** *algorithm?* (4) *Which embedding methods and clustering algorithms give the best results? What is the relation between the graph's properties and the way how it is embedded into the latent space?*

The rest of the paper is organized as follows. In Section 2 we further describe the proposed method and motivate it. Sections 3 and 4 introduce an experiment designed to test the hypothesis and, respectively, present obtained results. Finally, Section 5 provides some concluding remarks.

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 $\mathcal{A} =$

$\{A_1, A_2, \ldots, A_\ell\}$ of V that tries to maximize the modularity function $q_G(\mathcal{A})$, we perform the following three steps:

Step 1: Find the embedding function $\mathcal{E}\colon V \to \mathbb{R}^s$ which embeds each node of graph G into a s-dimensional latent 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 the partition $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. The goal is to use $\mathcal{C}$ 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 $\mathcal{A}$: $k \gg \ell$.
Step 3: Run the **Louvain** (or **Leiden**) algorithm on graph G using the partition $\mathcal{C}$ as a starting point. The result of this procedure, partition $\mathcal{A}$, is the outcome of our algorithm.

2.1 Motivation

First of all, let us discuss the reasons why one might want to use the embeddings at all. One issue is a nature of graphs as data structures; they are discrete objects, which reduces the number of possible approaches to the problem of community detection. It basically forces one to use the 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, mostly because there are more algorithms designed for working with real numbers and they are often more efficient [2]. Also, properly selected embeddings might be considered as a form of "denoising" data; they retain 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.

In the case of community detection algorithms these advantages are clearly visible. Instead of greedily merging nodes into communities, we merge them when that are close in the latent space, ensuring that the connections are more stable (not merely a result of a random enumerating). Indeed, equipped with a properly selected embedding, nodes that are close in the latent space will almost surely be part of the same community.

However, experimental results (see: [30]) show that using only an embedded representation to obtain the desired partition is not enough. An obvious explanation is a fact that embeddings are usually too reductive, that is, the representation gap between the 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 achieve a satisfactory result. Overcoming this issue is the main reason why the proposed solution consists of two separated partitioning steps. The reasoning is pretty straightforward: 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. Similarly, it is expected that these ideas will also

improve the quality of the results obtained by the **Leiden** algorithm—because of the additional refinement stage it gives a 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 clustering 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 the embedding $\mathcal{E}$ and selection of clustering algorithm. There are plenty of different embedding methods to choose from (see, for example, [6, 11, 9, 17]), 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 a guidance for future users. We also want to compare the results with divergence scores obtained by the **CGE** [15, 12]—unsupervised framework created to compare and asses different embeddings. We believe that this framework might become a useful tool, significantly simplifying the selection process of a suitable embedding.

Similarly, finding a clustering algorithm for the first step might be challenging. There are plenty of the well-known, efficient, and scalable algorithms; they might result in vastly different behaviour of the initial partitioning. For example, density-based algorithms will cluster only the points occupying the same densely connected regions whereas the points in the sparsely inhabited areas will be considered as noise and will not be assigned to any cluster. Thus, the initial partition C will contain only the nodes which are almost surely the parts of the same communities, leaving the more ambiguous nodes for the **Louvain** or **Leiden** algorithm. On the other hand, distribution-based methods of clustering will return the probability of a node belonging to each cluster, not a fixed assignment. As a result, one might fine-tune the certainty of the partition C instead of leaving it to the algorithm. Obviously, it is necessary to validate the described above intuitions which will be an important part of the experiment described in the next section.

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 GitHub repository[3], and so are Jupyter notebooks with a more details and further result analysis[4]. The experimental design was as follows. At the beginning, a comprehensive family of graphs with various properties was generated using the **ABCDe** (**A**rtificial **B**enchmark for **C**ommunity **D**etection) model [16, 13] and

[3] https://github.com/bartoszpankratz/ECCD

[4] https://github.com/bartoszpankratz/ECCD/blob/main/Embedding-Clustering_Community_Detection_Experiment.ipynb

following parameter sweep: the number of nodes $n = 1000$, 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 $\Delta = \sqrt{n}$ 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. Detailed explanation on how these parameters impact the graph structure is available in [16, 14, 13].

Louvain, **Leiden**, and **ECG** algorithms were each run 50 times for every given graph in order to obtain the baseline for the comparison. Then, every graph was embedded using the following algorithms taken from the Python *OpenNE*[5] package: **Locally Linear Embedding (LLE)** [29], **Laplacian Eigenmaps (LE)** [3], **deepWalk** [26], **node2vec** [10], **LINE** [31], **SDNE** [33], **GraRep** [7] and **HOPE** [24]. For each of the selected algorithms, we tested dimensions $d \in \{8, 16, 32, 64, 128, 256\}$. To find the most suitable clustering algorithm and get the best initial partitioning C, for every embedding $\mathcal{E}$ we tested the following three methods: **k-means** [20], **HDBSCAN** [22] and **Gaussian Mixture Model (GMM)** [28]. Parameters of all the embedding and clustering algorithms used in this experiment are further described in the aforementioned accompanying Jupyter notebook. Finally, every partition C was used as the initial partitioning for both **Leiden** and **Louvain** algorithm. To achieve comparable results, both methods were run 50 times on every C.

Roughly 55,000 different embeddings were tested with more than 1,500,000 initial partitions. Experiments were performed on the machines with 32 Intel Xeon Processors (Cascadelake) 2.30 GHZ vCPUs with 160GB RAM memory, 120GB disk space and Ubuntu 20.04.1 operating system. Computations were run simultaneously on eight machines for five consecutive days, totalling in around 960 vCPU hours.

4 Results

Figure 1 presents the results for one representative set of parameters: $\xi = 0.35$, $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$. As one can easily see, the results obtained by **Louvain** after using a better initialization procedure are clearly improved. Both **ECG** and **EC–Louvain**[6] are able to improve over the vanilla **Louvain**. In some rare cases, **EC–Louvain** is able to achieve performance similar to **Leiden**. But what is the most interesting, adding the initial partitioning C to **Leiden** significantly improves its quality and reduces the volatility.

The presented figure shows the results only for a single case; Table 1 shows how different values of ξ impact the performance of the algorithms. The relation here is pretty obvious; ξ is a *noise parameter*, it controls the expected fraction of edges between communities. As a result, with an increasing value of ξ one should

[5] https://github.com/thunlp/OpenNE

[6] **EC** stands for **Embedding–Clustering** and denotes the proposed extension of **Louvain** and **Leiden** algorithms. If not otherwise stated, the results for the **EC** algorithm uses the best possible initial partitioning C.

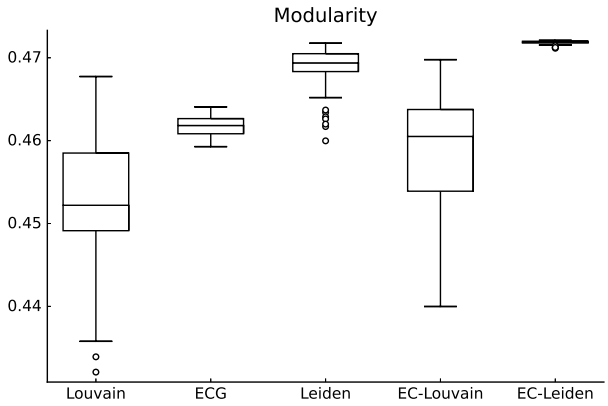

Fig. 1. Comparison of the modularity function for a single but representative set of parameters: $\xi = 0.5$, $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$.

expect the modularity to decrease, but also relative better performance of the augmented methods. We could see that **EC–Louvain** gives a relatively small improvement over the baseline **Louvain**, but **Leiden** with initial partitioning is able to outclass the rest of the algorithms with a large margin. Also it reduces the volatility to the negligible levels. Interestingly, for $\xi = 0.75$ **ECG** gives worse results than **Louvain**; **ECG** seems to be very sensitive to the graph's parametrization. In some cases it performs very well (see, for example, Figure 1), but it can also be weaker than **Louvain**. In comparison, it is never a case for the **EC** methods—in the worst case scenario, they return the same value of the modularity as **Louvain**.

Table 1. Comparison of the algorithms for different values of ξ ($\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$). Column *Louvain (baseline)* shows the average modularity obtained by this algorithm. Other columns present the average difference between the results of each algorithm and **Louvain**. Standard deviation is given in parenthesis.

ξ	Louvain (baseline)	ECG	Leiden	EC–Louvain	EC–Leiden
0.35	0.58132	0.00027	0.0029	0.00145	0.00302
	(0.00502)	(0.00019)	(0.00042)	(0.00237)	(0.0)
0.5	0.45263	0.00907	0.01593	0.00596	0.0192
	(0.00847)	(0.00124)	(0.00289)	(0.00696)	(0.0002)
0.75	0.30533	-0.01987	0.01955	0.00976	0.03096
	(0.00357)	(0.00279)	(0.0029)	(0.00448)	(0.00206)

One can see similar pattern for other parameters of the **ABCD** model[7]: when change of the parameter distorts the community structure of the graph, then the advantage from using the augmented methods is more visible. However, in almost all cases **EC**–**Louvain** gives small to mediocre improvement, but **EC**–**Leiden** gives a significant performance boost. Why is this happening? The answer is pretty straightforward and lies in the very nature of both algorithms, **Louvain** and **Leiden**.

As it was mentioned before, **Louvain** merges two nodes if such move maximizes the modularity locally, without any broader context. The initial partitioning C was designed to overcome this issue, guaranteeing the stability of the first step of the algorithm. But this problem is prevailing 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 procedure would be unfeasible for large graphs as it is very time consuming.

On the other hand, refinement stage in **Leiden** solves this issue. 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 the given subgraph induced by a single community. But still, **Leiden** backtracks only in a limited scope; early on, 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 C we ensure that this will not happen.

The last question remaining concerns the way how one should design the procedure. Clearly, proper selection of embedding and clustering gives a significant boost of the performance of **Leiden** (and to the lesser extend **Louvain**), but how should one chooses them?

Figure 2 shows the relation between the modularity and the CGE scores obtained by the unsupervised framework for comparing graph embeddings [15, 12], both local and global. Results show some interesting behavior. Let is first focus on the **EC**–**Louvain**. As can be seen on the two upper plots, the relation between the quality of the embedding and the achieved modularity is pretty insignificant, basically any kind of the reasonable embedding could give us a similar performance. These observations are in line with previous results showing that the inherent volatility of **Louvain** decreases the relevance of the initial partitioning C. However, it is not a case for **EC**–**Leiden**. We could clearly see that there is a strong relation between the quality of the embedding and the final modularity value. Moreover, plots show that **node2vec** is usually the best performing embedding algorithm. It is quite intuitive; it represents the nodes through the use of random walks and in the case of the community detection it is a natural form of representing proximities—nodes that are parts of the same

[7] For details please refer to: https://github.com/bartoszpankratz/ECCD/blob/main/Embedding-Clustering_Community_Detection_Experiment.ipynb

communities are likely to be present close to each other in the associated random walks.

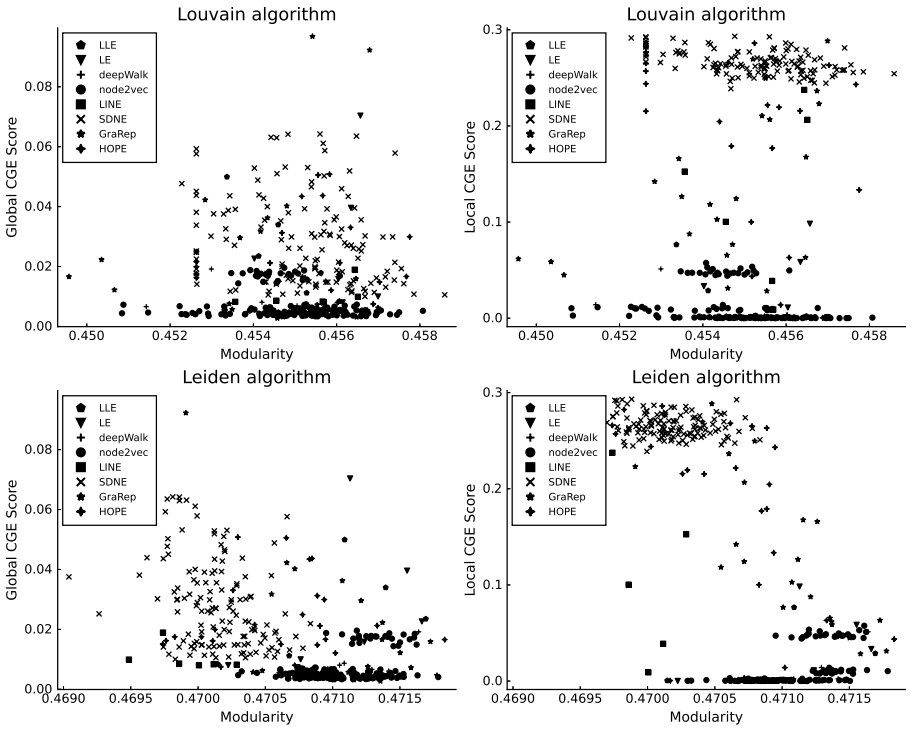

Fig. 2. Comparison of the modularity function and Global/Local CGE scores for different embedding algorithms and a single set of parameters: $\xi = 0.5$, $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$.

Let us now briefly comment on the performance of different clustering algorithms. **HDBSCAN** was usually the best one, which was somewhat foreseeable—this density based algorithm clusters nodes only when they are certainly a part of the same community. Figure 3 shows the result for an example but representative parametrization. Further analysis of the impact of parametrization of both embeddings and clustering algorithms is available in the accompanying Jupyter notebook.

5 Final Remarks

The results presented in this paper show that the usage of the initial partitioning C obtained by clustering of 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**

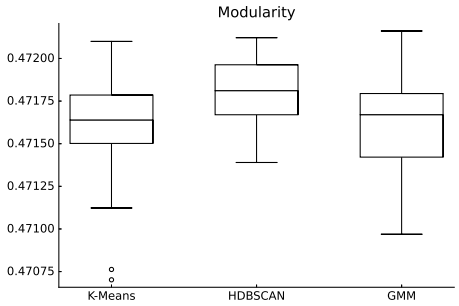

Fig. 3. Comparison of the modularity function for a graph embedded with **node2vec** into a 16–dimensional space and one set of parameters: $\xi = 0.5$, $\beta = 1.5$, $\gamma = 2.5$, and $\delta = 5$.

significantly improves its performance and reduce the volatility. We also provided results showing that there are some certain classes of embeddings (such as **node2vec**) and clustering algorithms (such as **HDB-SCAN**) that are the most suitable for this particular task.

6 Acknowledgments

Hardware used for the computations was provided by the SOSCIP consortium[8]. 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. Aiello, W., Chung, F., Lu, L.: A random graph model for massive graphs. In: Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing. p. 171–180. STOC '00, Association for Computing Machinery, New York, NY, USA (2000), https://doi.org/10.1145/335305.335326
2. Bartz-Beielstein, T., Zaefferer, M.: Model-based methods for continuous and discrete global optimization. Applied Soft Computing 55, 154–167 (2017), https://www.sciencedirect.com/science/article/pii/S1568494617300546

[8] https://www.soscip.org/

3. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic. p. 585–591. NIPS'01, MIT Press, Cambridge, MA, USA (2001)
4. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10), 10008 (2008)
5. Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE Transactions on Knowledge and Data Engineering 20(2), 172–188 (2008)
6. Cai, H., Zheng, V.W., Chang, K.C.C.: A comprehensive survey of graph embedding: Problems, techniques and applications (2018)
7. Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. p. 891–900. CIKM '15, Association for Computing Machinery, New York, NY, USA (2015), https://doi.org/10.1145/2806416.2806512
8. Fortunato, S.: Community detection in graphs. Physics Reports 486(3-5), 75–174 (feb 2010), https://doi.org/10.1016\%2Fj.physrep.2009.11.002
9. Goyal, P., Ferrara, E.: Graph embedding techniques, applications, and performance: A survey. Knowledge-Based Systems 151, 78–94 (Jul 2018), http://dx.doi.org/10.1016/j.knosys.2018.03.022
10. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks (2016)
11. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications (2018)
12. Kamiński, B., Kraiński, l., Prałat, P., Théberge, F.: A multi-purposed unsupervised framework for comparing embeddings of undirected and directed graphs (2021), https://arxiv.org/abs/2112.00075
13. Kamiński, B., Olczak, T., Pankratz, B., Prałat, P., Théberge, F.: Properties and performance of the abcde random graph model with community structure (2022), https://arxiv.org/abs/2203.14899
14. Kaminski, B., Pankratz, B., Pralat, P., Theberge, F.: Modularity of the abcd random graph model with community structure (2022), https://arxiv.org/abs/2203.01480
15. Kamiński, B., Prałat, P., Théberge, F.: An unsupervised framework for comparing graph embeddings. Journal of Complex Networks 8(5), cnz043 (2020)
16. Kamiński, B., Prałat, P., Théberge, F.: Artificial benchmark for community detection (abcd)—fast random graph model with community structure. Network Science pp. 1–26 (2021)
17. Kamiński, B., Prałat, P., Théberge, F.: Mining Complex Networks. Chapman and Hall/CRC (2021)
18. Lancichinetti, A., Fortunato, S.: Community detection algorithms: A comparative analysis. Physical Review E 80(5) (Nov 2009), http://dx.doi.org/10.1103/PhysRevE.80.056117
19. Leskovec, J., Lang, K.J., Mahoney, M.W.: Empirical comparison of algorithms for network community detection (2010), https://arxiv.org/abs/1004.3539
20. Lloyd, S.P.: Least squares quantization in pcm. IEEE Transactions on Information Theory 28, 129–137 (1982)
21. McCarthy, A.D., Chen, T., Ebner, S.: An exact no free lunch theorem for community detection. In: Complex Networks and Their Applications VIII, pp.

176–187. Springer International Publishing (nov 2019), https://doi.org/10.1007%2F978-3-030-36687-2_15
22. McInnes, L., Healy, J., Astels, S.: hdbscan: Hierarchical density based clustering. Journal of Open Source Software 2(11), 205 (2017), https://doi.org/10.21105/joss.00205
23. Newman, M.E.J.: Modularity and community structure in networks. Proceedings of the National Academy of Sciences 103(23), 8577–8582 (May 2006), http://dx.doi.org/10.1073/pnas.0601602103
24. Ou, M., Cui, P., Pei, J., Zhang, Z., Zhu, W.: Asymmetric transitivity preserving graph embedding. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 1105–1114. KDD '16, Association for Computing Machinery, New York, NY, USA (2016), https://doi.org/10.1145/2939672.2939751
25. Peel, L., Larremore, D.B., Clauset, A.: The ground truth about metadata and community detection in networks. Science Advances 3(5), e1602548 (May 2017), http://dx.doi.org/10.1126/sciadv.1602548
26. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (Aug 2014), http://dx.doi.org/10.1145/2623330.2623732
27. Poulin, V., Théberge, F.: Ensemble clustering for graphs: comparisons and applications. Applied Network Science 4(1) (Jul 2019), http://dx.doi.org/10.1007/s41109-019-0162-z
28. Rasmussen, C.E.: The infinite gaussian mixture model. In: Proceedings of the 12th International Conference on Neural Information Processing Systems. p. 554–560. NIPS'99, MIT Press, Cambridge, MA, USA (1999)
29. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science 290(5500), 2323–2326 (2000), https://science.sciencemag.org/content/290/5500/2323
30. Tandon, A., Albeshri, A., Thayananthan, V., Alhalabi, W., Radicchi, F., Fortunato, S.: Community detection in networks using graph embeddings. Phys. Rev. E 103, 022316 (Feb 2021), https://link.aps.org/doi/10.1103/PhysRevE.103.022316
31. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line. Proceedings of the 24th International Conference on World Wide Web (May 2015), http://dx.doi.org/10.1145/2736277.2741093
32. Traag, V., Waltman, L., van Eck, N.J.: From louvain to leiden: guaranteeing well-connected communities. Scientific Reports 9, 5233 (03 2019)
33. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1225–1234. ACM (2016)
34. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth (2012), https://arxiv.org/abs/1205.6233