CPTS 591: Elements of Network Science

Spring 2021

Semester Project Ideas

Last updated: April 1, 2021

General Introduction

This document describes some potential project ideas categorized under **five areas**. Some of the areas have multiple project ideas under them. In total, **12 project ideas** are described. Each project idea is described in its own section with its own set of references (or recommended readings). The project ideas are described in varying degrees of detail. If you select one of these project ideas to work on, you will work with me to develop it into a project proposal.

Project Area 1: Network Embedding

Background: Recently, network analysis methods that are based on representing networks in vector space, while preserving their properties, have become widely popular. Obtaining such an embedding is useful in a variety of applications including link prediction, clustering and visualization. An embedding is typically input as features to a model and the parameters are learned from training data. This obviates the need for complex classification models that are applied directly on the graph.

The use of embedding to study *network evolution* is an emerging research area. Evolutionary network analysis has gained increasing attention in recent years partly because more and more networks have characteristics that vary over time. For example, social networks, communication networks, and information networks continuously evolve over time, and it is desirable to learn how the network structure changes over time and if there are any other interesting trends over time in general. One challenging aspect of networks is that they are inherently resistant to parametric modeling that would allow us to express the edges in the network as functions of time. This is because, unlike multidimensional data, the edges in the network reflect interactions among nodes, and it is difficult to independently model an edge as a function of time, without taking into account its correlations and interactions with neighboring edges.

In the following set of project ideas, you are asked to propose graph embedding methods for temporal graphs using different techniques. You may use the embedded features to predict future changes on a graph or analyze current changes at the level of vertices.

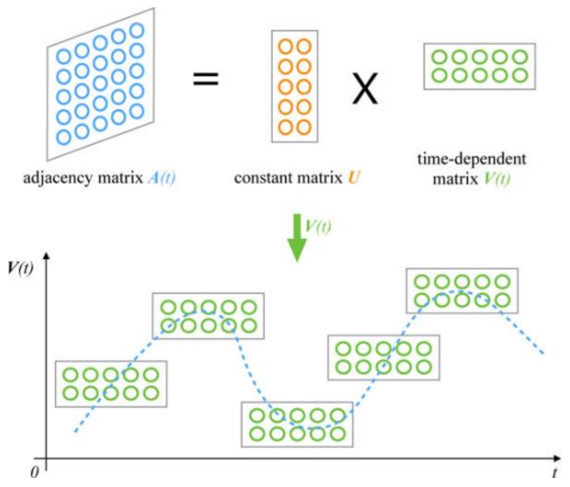
Project Idea 1.1: Factorization based Time-varying Network Embedding

For this project, you will need to use a matrix factorization method, in which the entries are parameterized by time. This approach allows us to represent the edge structure of the network purely as a function of time and predict the evolution of the network over time. The basic idea of temporal matrix factorization methods is to extract a low rank representation of an underlying adjacency matrix in a way that is parameterized with time. In the following formula, we have decomposed the adjacency matrix of a given graph into two lower rank matrices U and V.

$$\boldsymbol{A}(t) = \boldsymbol{U}\boldsymbol{V}(t)^T \text{ or } \boldsymbol{A}(t) = \boldsymbol{V}(t)\boldsymbol{V}(t)^T$$

The function V(t) can take on any canonical form, such as a linear model, a polynomial model, and so on. However, we are interested to keep the U matrix unchanged as time progresses. This way the matrix U maintains the intrinsic properties of the nodes, and the matrix V keeps track of their changes.

Challenge: How does one determine the values of U and V(t)? The standard approach in matrix factorization is to set up a least squares optimization problem, so that A(t) matches UV(t) as closely as possible. This can be achieved by minimizing the sum of the squares of the entries in A(t) − UV(t). Therefore, one can express this optimization problem as the minimization of the time-decayed sum of the matrix A(t) − UV (t) over all values of t from 1 to the current time CT.



Goals of the project: Tasks you need to accomplish in this project include the following. (1) Propose a general framework that decomposes a given graph stream to a lower ranked matrix U and V(t). (2) Evaluate the model using standard evaluation techniques (e.g. distance of the predicted and original matrices). (3) Test your model on at least two different datasets.

Additional Information:

There are several works in the literature that address this problem. The references [1, 2, 3, 4] give more details about the problem, its application and approaches for solving it. However, you are free to come up with your own optimization problem, cost function or even a new way to solve it.

References:

[1] Yu, Wenchao, Charu C. Aggarwal, and Wei Wang. "Temporally Factorized Network Modeling for Evolutionary Network Analysis." *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. ACM, 2017.

[2] Ahmed, Amr, et al. "Distributed large-scale natural graph factorization." Proceedings of the 22nd international conference on World Wide Web. ACM, 2013.

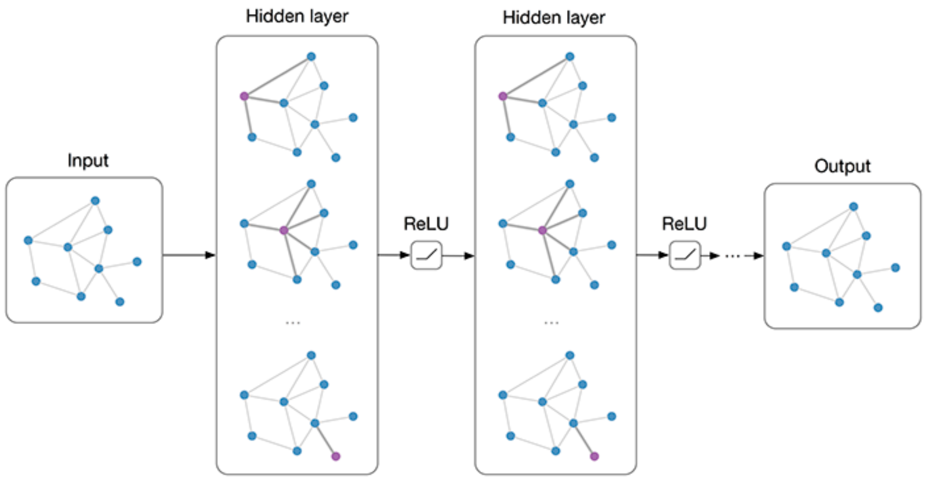
[3] Cao, Shaosheng, Wei Lu, and Qiongkai Xu. "Grarep: Learning graph representations with global structural information." Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. ACM, 2015.

[4] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In KDD, 2016

Project Idea 1.2: Neural Network based Time-varying Network Embedding

For this project idea, you will need to use a deep learning method to build a neural network framework to embed temporal features of each node of the graph. The basic idea of temporal matrix factorization methods is to extract a low rank representation of an underlying adjacency matrix in a way that is parameterized with time.

Challenge: What is the best way of modeling a graph structure in data to fit neural networks? One may use Graph Convolutional Network to capture features for each node. Also, it might be possible to use a recurrent neural network architecture to capture temporal changes and embed them to a low dimensional representation. What would be a good and representative loss function?



Goals of the project: (1) Propose a general framework that decomposes a given graph stream into a lower ranked matrix V of dimension n by d, where n is the number of nodes in the graph. (2) Evaluate the model using standard evaluation techniques (e.g. distance of the predicted and original matrices). (3) Test your model on at least two different datasets.

Additional Information:

There are several works in the literature that address this problem. In [1, 2], you can find more details about approaches proposing convolutional procedures on graphs. The codes associated with these papers are also available online, and you would need to extend them to capture the dynamic features of each node. However, you are free to come up with your own architecture, cost function or even a new way to solve the problem.

References:

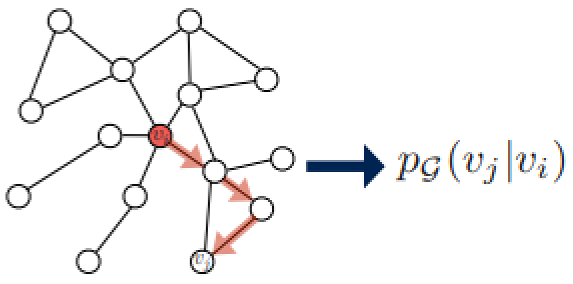
[1] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).

[2] Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering." Advances in Neural Information Processing Systems. 2016.

Project Idea 1:3: Random Walk based Time-varying Network Embedding

For this project idea, you will need to use a random walk method to embed changing features of each node in a graph. The basic idea behind random walk-based methods is to extract a low rank representation of the underlying adjacency matrix by defining a good and representative random walk in the graph that makes similar nodes close to each other. Then by using an embedding method, we can embed those features in such a way that the low dimensional features are a good representation of the node's behavior in comparison to others. A well-studied method to embed random walks is using language modeling methods such as word2vec.

Challenge: How does one define a random walk on a temporal graph that shows the changes of graph and be representative of different temporal behaviors? In the following figure, you see that starting from node $v_{i,}$ a random walk represents the neighbors and close by nodes, so it is possible to build a probabilistic model to capture the closeness of different nodes in a graph. But what happens if each edge has a timestamp on it? How can those random walks be used to tell the story of graph changes and build time-related probabilities?



1. Run random walks to obtain co-occurrence statistics.

Goals of the project: (1) Propose a general framework that decomposes a given graph stream to a lower ranked matrix V (n by d), where n is the number of nodes in the graph. (2) Evaluate the model using standard evaluation techniques. (e.g. distance of the predicted and original matrices). (3) Test your model on at least two different datasets.

Additional Information: There are several works in the literature that address the problem of graph embedding using random walks. The references [1, 2, 3] provide more details about this problem, its applications and approaches taken to solve it. However, you are free to come up with your own random walk procedure, embedding, and optimization problem to solve it.

References:

[1] Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014.

[2] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.

[3] Bamler, Robert, and Stephan Mandt. "Dynamic word embeddings." International Conference on Machine Learning. 2017.

Datasets for Project Ideas 1.1—1.3:

Below are example of datasets you may find useful to build and evaluate your model.

- Digg: This is the reply network of the social news website Digg. Each node in the network is a user of the website, and each directed edge denotes that a user replied to another user. Link: http://konect.uni-koblenz.de/networks/munmun_digg_reply
- Enron-mail: The Enron email network consists of 1,148,072 emails sent between employees of Enron between 1999 and 2003. Nodes in the network are individual employees and edges are individual emails. It is possible to send an email to oneself, and thus this network contains loops. Link: http://konect.uni-koblenz.de/networks/enron
- Other dynamic networks (with timestamp) are available in this link: http://konect.uni-koblenz.de/networks/

Project Area 2: Core-Periphery Structures

Background: Some complex networks can have features that may not be revealed by studying nodes and their neighborhoods "locally". By studying the network structure "globally" instead, it is possible to understand the topological construction of the network, and relationships and interactions between constituent parts. For example, it is very common to see strong inter-connections between a certain number of nodes (they form the core), and some nodes very weakly linked to the nodes belonging to the core (they form the periphery).

We envision two main categories of project ideas under this project area.

Project Idea 2.1 - theoretical, algorithm development for identifying core – periphery structure of generic complex networks.

Project Idea 2.2 - application oriented: Core-periphery structure has been used to study anomalies in network structures, protein molecule interactions, formation of social and economic groups, disconnect between international airports and domestic airports.

<u>Recommended Reading Papers:</u>

<u>For Project Idea 2.1:</u>

- Core-Periphery Structure in Networks - Rombach, et al https://arxiv.org/pdf/1202.2684.pdf
- A Unified Method of Detecting Core-Periphery Structure and Community Structure in Networks - Xiang et al. **https://arxiv.org/abs/1612.01704**
- Core-periphery clustering and collaboration networks - Crecenzi et al. http://ieeexplore.ieee.org/document/7752285/

<u>For Project Idea 2.2:</u>

- Core Periphery Structures in Weighted Graphs Using Greedy Growth - Sardana et al. http://ieeexplore.ieee.org/document/7817029/

<u>For both project idea categories in a single paper:</u>

- Finding multiple core-periphery pairs in networks - Kojaku et al https://arxiv.org/abs/1702.06903
- A Graph Modification Approach for Finding Core–Periphery Structures in Protein Interaction Networks - Bruckner et al. https://link.springer.com/chapter/10.1007/978-3-662-44753-6_25

Some Datasets and More Specific Project Ideas (under Category 2.2):

Project Idea 2.2.1:Predictive toxicology.
Data on chemical interaction to predict whether introduction of certain toxic substance in rats cause cancer in the long run. Can we identify strong interconnections between certain chemicals that form the core, and some that are only weakly linked? Our conjecture is - core-periphery analytics can help identify cancer-causing interactions which might get filtered out in clustering kind of analysis.
https://www.predictive-toxicology.org/ptc/#InitData

- *Original coding required: high*
- *Data cleaning required: high*
- *Publication/Future work potential: high*

Project Idea 2.2.2: Economy Prediction.
Which regions in US (or other counties) are more tightly connected to one another, and can be used as predictors of economy? Basically, US road network data is available here: http://www.dis.uniroma1.it/challenge9/download.shtml US Economy data is available here: https://catalog.data.gov/dataset?organization=doc-gov&tags=gdp

- *Original coding required = high*
- *Data cleaning required = low for road data, high for economic data*
- *Publication/Future work potential = medium*

Project Idea 2.2.3: Bitcoin data.
http://snap.stanford.edu/data/soc-sign-bitcoinotc.html
It is important to maintain a who-trusts-whom network for block-chain based technologies such as bitcoins. Core-periphery structures might help in detection of anomalies in transactions, or observe network-level transaction trends in crypto-currencies.

- *Original coding required = low*
- *Data cleaning required = low*
- *Publication/Future work potential = high*

Project Area 3: Graph-based semi-supervised learning

Background: In many real-life machine learning tasks, labeled data is scarce, whereas unlabeled data is easily available. Active semi-supervised learning is an effective approach for such scenarios. A semi-supervised learning technique must not only learn from the labeled data but also from the inherent clustering present in the unlabeled data. Applying a graph perspective to semi-supervised learning has been found fruitful. In a graph-based formulation, the data points are represented by nodes of a graph and the edges capture the similarity between the nodes they connect. For example, the weight on an edge might be a function of the distance between the two points in the feature space chosen for the classification task.

Project Idea 3.1: For this project, we look at the problem from a "graph signal" point of view. A graph signal is defined as a scalar-valued discrete mapping $f : V \rightarrow R$, such that $f(i)$ is the value of the signal on node i. The membership function of a given class can be thought of as a graph signal, which has a scalar value at each of the nodes (e.g., 1 or 0 depending on whether the data point belongs to the class). Since features have been chosen to be meaningful for the classification task, it is reasonable to expect that nodes that are close together in the feature space will be likely to have the same label. Conversely, nodes that are far away in the feature space are less likely to have the same label. Thus, we expect the membership function to be smooth on the graph, i.e., moving from a node to its neighbors in the graph is unlikely to lead to changes in the membership. Thus, the semi-supervised learning problem can be viewed as a problem of interpolating a smooth graph signal.

You are asked to leverage existing works on graph signal sampling and interpolation (sampling theory for graph signals); these works provide a rigorous and unified framework to select points to be labeled and subsequently perform semi-supervised learning. The framework we envision will provide conditions under which a graph signal can be uniquely recovered from its values on a subset of vertices.

For this project, you will be given access to two different labeled datasets for health applications and you are asked to apply a previously proposed approach on these datasets. For this purpose, you will carry out the following steps (and others as needed):

1) Build a graph for each dataset: in this step, you will extract meaningful features from each data sample and then measure the similarity of different samples using an appropriate method (e.g. cosine similarity).
2) Use an available greedy approach [1] to find the minimum set of nodes to predict the graph signals (i.e. labels for each node) for the entire network (the code is available online).
3) Measure the accuracy of prediction by comparing it with the ground truth label for each data sample in each dataset.

References:

[1] *Akshay Gadde, Aamir Anis, and Antonio Ortega.* "Active semi-supervised learning using sampling theory for graph signals." *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014.

[2] *Aamir Anis, Akshay Gadde, and Antonio Ortega.* "Towards a sampling theorem for signals on arbitrary graphs." *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014.

[3] *Narang, Sunil K., et al.* "Localized iterative methods for interpolation in graph structured data." *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*. IEEE, 2013.

Project Area 4: Sequence Similarity Networks and Distance Metrics

Two project ideas are described under this area. There is one set of references (shared by the two ideas) at the end of the section.

Project Idea 4:1: Sequence Similarity Networks

Sequence similarity networks are a common way of modeling relationships between biological sequences, particularly proteins [1]. They are useful for both visualization and analysis such as clustering and centrality analysis. Typically these networks assume some distance metric between strings. The metric is then used to define a threshold for which a network can be constructed. In particular, if two sequences are more similar than the threshold, they are connected by a (weighted) edge; if their similarity falls below the threshold, the edge is omitted. However, this approach has several limitations. First, it is susceptible to error when distance metric is approximate (which is common, due to the time complexity of exact distance computation between strings). More fundamentally, appropriate thresholds are difficult to determine in practice. A threshold set too high leads to an intractably dense network, while one set too low leads to loss of meaningful structure. Alternative network construction techniques, such as nearest neighbor networks [2], exist, and in particular, the Directed Weighted All Nearest Neighbors [3] variant can mitigate the need for such a threshold.

For this project you are encouraged to apply the DiWANN network model to one or more text dataset(s) (which may or may not be biological in nature), and compare its structure against other network models. Use a single distance measure across all models, although you may choose whichever you find most appropriate. Design experiments to demonstrate the effects of different network construction methods on your dataset, and show how the network structure is altered (or maintained). Features to consider may include degree distribution, connected components, clustering, centrality etc.

Project Idea 4:2: String Distance Metrics

Unlike numerical and categorical features, which have a variety of distance metrics (Euclidian, Jaccard, cosine, etc) that can be computed at a low time complexity, finding the distance between two pieces of text (i.e. strings) is computationally expensive. The ideal standard is Levenshtein distance (or edit distance) [4] or alignment based scores (depending on the underlying assumptions about the given strings), but these methods rely on dynamic programming, and thus take $O(|S1||S2|)$ time [5], where S1 and S2 are the two strings being compared. For long strings this can quickly become infeasible, particularly in applications where many distance calculations are required (for example generating a distance matrix for a similarity network).

To alleviate this problem some approximations have been developed. One method is to use approximate string searching techniques such as BLAST [6] to generate a heuristic for distance. However these scores have been shown to correspond to edit distance poorly in some cases. Other novel methods have been developed as well, some of which use derived features such as character counts to approximate distance [7]. Another technique is to represent a string as a time series where each new character either increments or decrements the value by a certain amount and each character is represented by a new timestep [8]. With this representation, distance between two time series' can be computed with relative ease. This method is robust to shifts and gaps, however, it is also sensitive to alphabet size and ordering insofar as the distance between two strings will depend on which characters are assigned which values, and a single character edit will change the distance by different amounts depending on a particular edit (changing an 'a' to an 'f' will have a different effect than changing it to a 'y'). The issues are exacerbated as alphabet sizes increase, so the method can be useful for strings with a small alphabet (such as DNA), but less so for longer alphabets (amino acids, natural language, etc.).

For this project, you are encouraged to research string distance metrics, or develop your own, with the goal of constructing a similarity network in mind. You are especially encouraged to consider the time series variant described in [8]. The metric(s) you investigate may be approximate, exact or heuristic, and may rely either on only the original string, or on some set of features producible in an efficient manner. Compare these method(s) in terms of robustness, time/space to compute and accuracy (if inexact) to other known methods. Outline the strengths and weaknesses of your approach and design experiments to demonstrate this on some real world dataset.

Available Real-world Dataset: Short Sequence Repeats

We describe here a dataset that we have in our lab (SCADS) and that can be used for the two project ideas described above (although if you use it for the first project, you will need to supplement it with some additional dataset). You will be provided access to this dataset. However, if you would like to work with some other dataset for these two project ideas, you are welcome to do so (but first discuss it with me).

Short sequence DNA repeats (SSRs) are relatively short (<500 base pair) patterns that recur (inexactly) in the genes of many species. These repeat patterns are used to characterize strains of some microbes. Typically this is done by naming each version of an inexact repeat and characterizing a strain as the pattern of those repeats that it contains. So if the repeat section of a given gene were "aagtat-acgtat" (note that the dashes are for readability only) we might call aagtat R1 and acgtat R2, making the prior strain the R1R2 strain. If another sample has the same pattern, we could be relatively sure they were closely related, whereas a new strain with the sequence "aagtat-acgtat-aagtat" could now be known as the R1R2R1 strain.

We have a dataset on SSR data for *Anaplasma marginale*, a pathogen which commonly infects cattle. In some cases, base pairs are replaced by the amino acids they encode (3 basepairs to 1 amino acid, deterministically). For more general information on SSRs and this type of genotyping, refer to this paper (https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-016-2686-2)

From a data perspective, you can safely assume that one character of a given repeat in this data set is a single amino acid. The data set contains the following: the set of all known repeats, which are one or more names and an amino acid sequence; the set of all known strains, which have zero or more names, a sequence of repeats, and a set of locations (country[, province][, county/municipality] where bracketed fields are optional) where they have been reported. The formats of the two files are as follows :

	Repeat Name	Repeat Sequence
1	A	ddsssasgqqqessvssqseastssqlg
2	B	adsssaggqqqessvssqsdqastssqlg
3	25	adsssaggqqqessvssqsgqastssqlg
4	α	adsssasgqqqessvssqseastssqlgg
…		

	Strain Sequence	Location 1	Location 2	…	Location N
1	A A B α	27.6648274, -81.5157535	NA	…	NA
2	α α 25 B A	-18.512178, -44.5550308	27.6648274, -81.5157535	…	41.87194, 12.56738
…					

References

1. Atkinson HJ, Morris JH, Ferrin TE, Babbitt PC. Using sequence similarity networks for visualization of relationships across diverse protein superfamilies. PLoS One. 2009;4.

2. Eppstein D, Paterson MS, Yao FF. On nearest neighbor graphs. Discrete Comput. Geom. [Internet]. 1997;17:263–82. Available from: http://www.springerlink.com/index/10.1007/PL00009293

3. Catanese HN, Brayton KA, Gebremedhin AH. A nearest-neighbors network model for sequence data reveals new insight into genotype distribution of a pathogen. BMC Bioinformatics. 2018;19.

4. Levenshtein VI. Binary codes capable of correcting spurious insertions and deletions of ones. Sov. Phys. Dokl. 1966;10:707.

5. Waterman MS, Smith TF, Beyer WA. Some biological sequence metrics. Adv. Math. (N. Y). 1976;20:367–87.

6. Camacho C, Coulouris G, Avagyan V, Ma N, Papadopoulos J, Bealer K, et al. BLAST+: architecture and applications. BMC Bioinformatics. 2009;10:421.

7. Seker SE, Altun O, Ayan U, Mert C. A Novel String Distance Function Based on Most Frequent K Characters. Int. J. Mach. Learn. Comput. [Internet]. 2014;4:177–82. Available from: http://www.ijmlc.org/index.php?m=content&c=index&a=show&catid=44&id=446

8. Shieh J, Keogh E. iSAX: Indexing and Mining Terabyte Sized Time Series. 14th ACM SIGKDD Int. Conf. Knowl. Discov. data Min. [Internet]. New York, NY, USA: ACM; 2008. p. 623. Available from: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.155.4531 http://dl.acm.org/citation.cfm?doid=1401890.1401966%5Cnhttp://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.155.4531%5Cnhttp://dl.acm.org/citation.cfm?doid=1401890.1401966

Project Area 5: Synthetic Network Generation with Multiple Constraints

Background: Three project ideas are described under this area. There is a set of references at the end of the section that is related to the problem and the project ideas we discuss. All three of the project ideas deal with generating a directed synthetic network with arbitrary number of nodes that has similar structural characteristics as that of a given real-world network. The characteristics could be in/out degree distribution, clustering coefficients, node centralities, and/or other parameters that define a structure of the network. The problem of generating synthetic yet realistic networks is a very important problem when it comes to simulating the behavior of the network under different scenarios when we model a real-world problem with graphs. What we are modeling could be biological network, web traffic, hospital transfer networks, web social media etc.

The challenge here is that in directed graphs, each edge between two nodes can affect several metrics in the network being generated, which makes the network generation problem an optimization problem in which our goal is to minimize the distance between the characteristics metrics in the given real-world graph and in the synthetic one.

Examples of suggested metrics to compare between the given real-world network and the generated synthetic network include:

- S1: In-degree distribution
- S2: Out-degree distribution
- S3: The distribution of sizes of weakly connected components ("wcc")
- S4: The distribution of sizes of strongly connected components ("scc")
- S5: Hop-plot: the number P(h) of reachable pairs of nodes at distance h or less; h is the number of hops.
- S6: Hop-plot on the largest WCC.
- S7: The distribution of clustering coefficient

In the following, you may find a few rough ideas on how you may approach this problem. These are all rough ideas but each of them is worth exploring:

Project Idea 5:1: Generative Adversarial Networks Approach

Generative Adversarial Networks or GANs are well-known in the deep learning area for building models that are capable of generating realistic datasets by trying to find and deliver the most important characteristics of the real data. Since the nature of this network generation problem is basically the same, GAN models look like a natural fit. There are some previous works that have tried to solves this problem with GAN models but none of them addressed the directed graph network with arbitrary number of nodes and the structural metrics like degree distribution as we described earlier.

Project Idea 5:2: Graph Laplacian Scaling

As you know, the Laplacian matrix of a graph embeds different characteristics of the graph into the nodes. You can deduce the graph modularity, different levels of graph and subgraph clustering and several other characteristics of the graph by looking at their eigen-values and vectors. Resampling the matrix in a correct way may be a way to scale the whole network by keeping these features of the graph and one can translate the scaled network to its original adjacency network with new number of nodes.

Project Idea 5:3: Graph Fourier Transform

Graph signal processing is an area of research where we treat graphs and the characteristics on their nodes as signals and use the conventional Fourier transform methods to model them as a function in frequency domain. This transformation enables one to analyze the coefficients of a function and the function helps to recreate the original signal by inverting the transformation. If the first transformation keeps the most important structural characteristics that we are looking for, there should be a way to resample the number of nodes we want in the frequency domain and generate a completely new graph with different number of nodes but same characteristics.

References

[1] Bojchevski, Aleksandar, et al. "Netgan: Generating graphs via random walks." *arXiv preprint arXiv:1803.00816* (2018).

[2] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.

[3] S. Chen, R. Varma, A. Sandryhaila and J. Kovačević, "Discrete Signal Processing on Graphs: Sampling Theory," in IEEE Transactions on Signal Processing, vol. 63, no. 24, pp. 6510-6523, Dec.15, 2015.

[4] A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura and P. Vandergheynst, "Graph Signal Processing: Overview, Challenges, and Applications," in Proceedings of the IEEE, vol. 106, no. 5, pp. 808-828, May 2018.

[5] Kolda, Tamara G., et al. "A scalable generative graph model with community structure." SIAM Journal on Scientific Computing 36.5 (2014): C424-C452.

[6] Leskovec, Jure, and Christos Faloutsos. "Sampling from large graphs." *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006.

[7] Newman, Mark EJ, Steven H. Strogatz, and Duncan J. Watts. "Random graphs with arbitrary degree distributions and their applications." Physical review E 64.2 (2001): 026118.