Mini-Project on Embeddings of Graphs

based on the article

Sparse Partitions by B. Awerbuch and D. Peleg

Mini-Project by: Eyal Shagrir

Advisor: Prof. Michael Elkin

Introduction

This Mini-Project is based on the article "Sparse Partitions", written by Baruch Awerbuch and David Peleg, and published in *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science, 1990, pp. 503--513*.

The main purpose of this article is to present techniques to represent arbitrary networks in an efficient and simple way. The solutions described in the paper are especially applicative in the area of distributed network algorithms, which require a qualitative representation of large scaled arbitrary networks.

In this manner, networks are simply graphs that may represent objects by nodes, and connections by edges, while the weights of the edges indicate the nature of the relationship between the objects.

Therefore, if we have a network $G = (V, E)$, we seek for some representation of it, that will allow us to maintain and update the information it holds efficiently, without saving and searching the entire graph data structure all along.

The representation dealt with in this article is called ***clustered representation of networks***. The idea behind it is to maintain a collection of clusters, i.e. groups of connected nodes in graph G, which covers the vertices set V . In this way, a large network could be "broken" into several components, that on the one hand cover the entire data set, and on the other hand is relatively easy and convenient to work with.

Awerbuch and Peleg discuss two basic principles for evaluating a qualitative cluster cover:

1. Clusters "size"
2. Cover sparsity

The first criteria, cluster "size", is related to the level of interaction between the nodes in each cluster, i.e. "internal communication". Intuitively, the larger a size of a cluster is, the harder it is to communicate for the nodes inside it.

The second one, cover sparsity, concerns the level of interaction between each cluster in the collection, i.e. "external communication". Also Intuitively, the sparser the cover is, the harder it is to communicate for the clusters in the collection.

Practically, cluster "size" is described by the collection's **radius**, and the cover sparsity by the collection's **degree**.

When seeking for a qualitative cluster cover, **we aspire for small radius and degree values**. This is because we want to minimize the computational complexity of interaction inside each cluster, and in addition to that, minimize the "external communication" between the clusters, as much as possible, so every procedure executed in the network, will be performed in each cluster separately.

Following that, the article presents an algorithm, "MAX COVER" (and its subroutine "Procedure Cover"), that given a cluster cover S to graph G, and an integer $k \geq 1$, it constructs a coarsening cluster cover T, that takes into consideration the above-mentioned two principles, by upper-bounding its radius and degree. As well as S, T also satisfies full nodes coverage to G, by the definition of coarsening.

In practice "MAX COVER" guarantees the following properties to the coarsening cover T, with respect to the given cover S, and constant k:

1. $\boldsymbol{Rad(T) \leq (2k-1) \cdot Rad(S)}$
2. $\boldsymbol{Deg(T) \leq 2k \cdot |S|^{\frac{1}{k}}}$

Important Definitions:

Given an undirected weighted graph $G = (V, E)$

Covers and clusters:

- $S \text{ is a } \boldsymbol{cover} \text{ of } G \boldsymbol{ iff } \bigcup_{s \in S} s = V$
- $\text{let } S, T \text{ be covers of } G. T \boldsymbol{ coarsens } S \boldsymbol{ iff } \text{for every } s \in S \text{ there exists } t \in T \text{ such that } s \subseteq t$
- $C \text{ is a } \boldsymbol{cluster} \text{ in } G \boldsymbol{ iff } C \subseteq V \text{ and the induced graph } G(C) \text{ is connected}$
- $S \text{ is a } \boldsymbol{cluster\ cover} \text{ of } G \boldsymbol{ iff } S \text{ is a cover of } G \text{ and for each } s \in S \ s \text{ is a cluster in } G$

From now on cover will denote cluster cover

Cover Radius:

- $\text{let } u, v \in V, \text{then } \boldsymbol{dist}(\boldsymbol{u}, \boldsymbol{v}) \text{ is the (weighted) length of the shortest path between } u \text{ and } v \text{ in } G$
- $\text{let } v \in V, \text{then } \boldsymbol{Rad}(\boldsymbol{v}) = \max_{w \in V}\big(dist(v, w)\big)$
- $\boldsymbol{Rad}(\boldsymbol{G}) = \min_{v \in V} Rad(v), \text{also known as the } \boldsymbol{radius\ of\ graph\ G}$
- $\text{let } C \text{ be a cluster } in\ G, \text{then } \boldsymbol{Rad}(\boldsymbol{C}) \text{ denotes } Rad\big(G(C)\big), i.e. \text{the radius of the induced graph by } C$
- $\text{let } S \text{ be a cover of } G, \text{ then } \boldsymbol{Rad}(\boldsymbol{S}) = \max_{C \in S}\big(Rad(C)\big), \text{also known as the } \boldsymbol{radius\ of\ cover\ S}$

Cover Degree:

- $let\ v \in V\ and\ S\ a\ cover\ of\ G, then\ \boldsymbol{Deg(v,S)} = |\{C \mid C \in S\ and\ v \in C\}|$
- $let\ S\ be\ a\ cover\ of\ G, then\ \boldsymbol{Deg(S)} = \max_{v \in V}\big(Deg(v,S)\big)$, $also\ known\ as\ the\ \boldsymbol{degree\ of\ cover\ S}$

The Algorithm

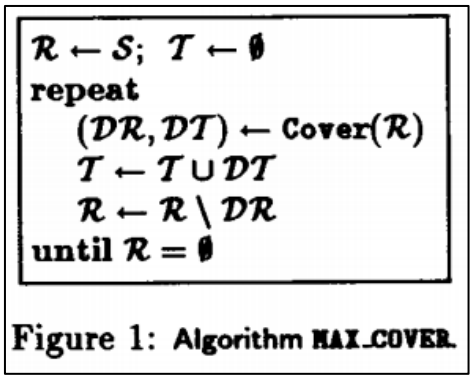


Figure 1: Algorithm MAX_COVER

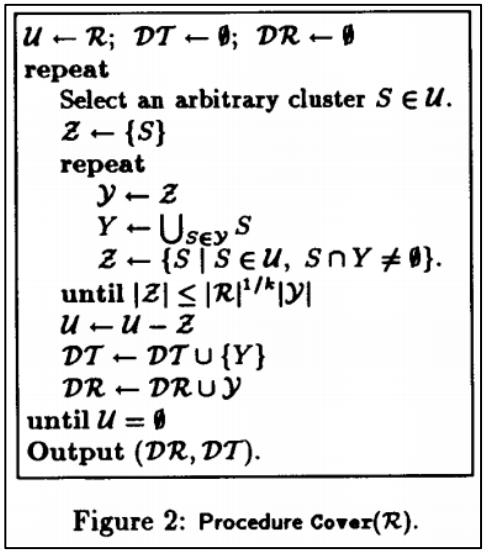


Figure 2: Procedure Cover($\mathcal{R}$).

The algorithm as described in figures 1,2 above is composed by the main algoritm, named "MAX_COVER" and its subroutine "Procedure Cover".

As mentioned before, given a cover S to graph G and integer $k \geq 1$ "MAX_COVER" algorithm constructs a new cover to G, T, that coarsens cover S and has the following properties:

1. $Rad(T) \leq (2k-1) \cdot Rad(S)$
2. $Deg(T) \leq 2k \cdot |S|^{\frac{1}{k}}$

We first discuss the subroutine "Procedure Cover" and its contribution to the algorithm.

"Procedure Cover" gets as its input, a collection of possibly overlapping clusters R in graph $G = (V, E)$, and a constant integer $k \geq 1$. Its output is two clusters collections, DR and DT which satisfy the following properties:

1. $DT\ coarsens\ DR$
2. $Y \cap Y' = \emptyset\ for\ every\ Y, Y' \in DT$
3. $|DR| \geq |R|^{1-\frac{1}{k}}$
4. $Rad(DT) \leq (2k-1) \cdot Rad(R)$

In practice, "Procedure Cover" works in two nested loops. In the outer loop, it first chooses a random cluster s from the collection and immediately enters the inner loop with a set $Z = \{s\}$.

In the inner loop, it begins by constructing a copy of Z, $y = Z$, and another set $Y = \bigcup_{s \in y} s$, which is practically the union of all the current clusters in y. Then, it adds to Z all the clusters from the original collection, which intersect with Y, i.e. all the clusters in R that share the same elements as some clusters that currently belong to Z. It does that until $|Z| \leq |R|^{\frac{1}{k}} \cdot |y|$ and then exists the inner loop.

It continues the outer loop by removing the clusters collected in Z from the original collection R, adding to DT the set $\{Y\}$, which is again the union of all clusters in y, and adding to DR the set y which holds the clusters from R before expanding Z to include the clusters that intersect with Y.

The outer loop is iterated until R is empty, which means all clusters were taken into consideration in some iteration by the inner loop.

Although the properties "Procedure Cover" provides to DR, DT are relatively hard to infer intuitively without a formal proof, we can see in the algorithm some main points that can help us learn about the attributes of DR and DT. For instance, at the end of the outer loop, we add $\{Y\}$ to DT and y to DR, and as mentioned above, $Y = \bigcup_{s \in y} s$. This implies directly the correctness of property 1: $DT\ coarsens\ DR$.

It is important to mention for the future, and more specifically for the linking of "Procedure Cover" and "MAX COVER", that "Procedure Cover" doesn't work directly on its input collection R, but on a copy of it.

We can now discuss the main algorithm, "MAX COVER", which is actually pretty simple and straight forward. It contains one loop, which in each iteration calls "Procedure Cover" with a copy of the original cover $R = S$, and returns two collection DR, DT with the four properties mentioned before. Then, it adds DT to the output cover T, and removes from R all the clusters that belong to DR. It does that until R is empty.

Again, the two properties guaranteed by "MAX COVER" regarding the output T are not deduced directly from looking at the pseudo-code of the algorithm, but we can see some relations between "Procedure Cover"'s output and "MAX COVER"'s. For example, the fact that DT coarsens DR, and then accordingly T coarsens S, or the fact $Rad(DT) \leq (2k-1) \cdot Rad(R)$ and then accordingly

$Rad(T) \leq\ (2k-1) \cdot Rad(S)$.

The Algorithm Implementation

This Mini-Project is written entirely in Python. Therefore, the implementation of the algorithm itself is very similar to the pseudo-code presented in the article, thanks to Python's built-in data structures: 'set' and 'frozenset'. The 'set' data structure allows maintaining a collection of elements very easily, and performing set operations like union and intersection with appropriate built-in Python 'set' functions. In this manner, the 'frozenset' data structure is exactly the same as 'set', except its elements are immutable. Hence, a collection of clusters, which is practically a set of sets, is represented in the implementation, as a set of frozen sets.

Nonetheless, in order to provide a convenient environment to the algorithm, there was a need to implement basic, yet complex objects and functions. Among some those were:

- Generating weighted, connected random graphs in different sizes and edge creation probability.
- Generating random clusters in graph in different sizes.
- Generating a random clusters cover to a given graph, also with varying cluster and cover size.
- Calculating the radius of a graph and the radius of its cover.
- Calculating the degree of a cover.

All of the above were written using a special Python graph library, called "networkx", and of course the built-in library "numpy" for math calculations. However, some graph operations were not supported at all by the "networkx" library, and required a more detailed implementation, especially the operations involved with clustering. Thus, we will discuss the implementation of the functions that generate random clusters and random clusters covers, more thoroughly:

- Generating a random cluster in graph:

  The function in charge of this operation, gets as optional parameters an upper bound and a lower bound for the cluster size. Otherwise, the lower bound is 1 and the upper bound is n, i.e. the number of nodes in the graph.

  It first chooses randomly the size of the cluster, between the lower bound and the upper bound. Then, it chooses randomly a node u in the graph, to be the first node in the cluster. It then enters a loop, which in each of its iterations, it chooses a random node u from within

the current cluster, and some other random node v from the graph, which is a neighbor of u, and adds it to the cluster. This loop continues until the cluster is in the predetermined size.

- Generating a random clusters cover in graph:
  The function in charge of this operation, gets as a parameter the **minimum size of the cover**, i.e. the minimum number of clusters it will contain. As before, it also gets as optional parameters an upper bound and a lower bound for the size of the clusters inside it.
  Following that, the function enters a loop, which in each of its iterations, it generates a random cluster with the cluster size bounds given as input, and adds it to the cover set. The loop continues to iterate, until the clusters cover indeed covers the nodes of the graph, **and** the size of the cover is at least as big as the cover size parameter. This means that the size of the returned cover, might be larger than the cover size parameter given to the function.

Note that the correctness of all of the utility functions and objects, and the correctness of the algorithm itself (i.e. the properties each procedure satisfies), were tested with Python built-in “pytest” framework.

Experiments

The experiments performed in this Mini-Project aspire to examine the effects of different graph attributes and algorithm parameters, on the coarsening cover resulting from "MAX COVER". With respect to the output's properties the algorithm guarantees, we measure these effects, by observing the radius and degree of covers.

Thus, we execute three experiments that analyze the effects of three key elements in the algorithm:

1. Graph Density Experiment:
   We seek a connection between the density of a graph, and the radius and degree of random covers generated in it, and the radius and degree of their coarsening covers.
   We do it by gradually increasing the edge creation probability, when generating a random graph. For each random graph, we generate some random covers, calculate their radius and degree, and then run "MAX COVER" on these covers, and calculate their radius and degree too.
   We present the radius and degree of the original cover and the coarsening cover, as a function of the edge creation probability, which represents the graph's density.

2. Cover Size Experiment:
   We wish to examine the effect of different sizes of graph covers, which differ by radius and degree themselves, on the resulting coarsening covers properties in accordance.
   In order to do that, we gradually increase the size of covers for each graph tested in the experiment, and then again, calculate their radius and degree, and the same for their coarsening covers, outputted from "MAX COVER".
   We present the radius and degree of the original cover and the coarsening cover, as a function of the cover size.

3. K Integer Experiment:
   Here, we want to examine an algorithm parameter, given to it as an input, and its impact on the results. Therefore, we generate some random graphs and covers, and for each cover we call "MAX COVER" with a different gradually increasing k. As before, we calculate the radius and degree of the original and coarsening covers.

Similarly, we present the radius and degree of the original cover and the coarsening cover, as a function of the parameter k.

Note that in this case, the original cover's radius and degree don't depend on the varying experiment element, k. Therefore their graphs are constant, representing the average radius and degree of the original covers in the experiment.

Experiments Implementation

We first present the general experiment structure, and then address each type specifically.

General Experiment Structure:

All the graphs that participate in the experiments are undirected, weighted and connected random graphs. More specifically, all of the graphs have 100 nodes and random edges weights between 0 and 100 (only integers).

Note that calculating a graph radius is an expensive operation, which takes most of the time when performing the experiments. Hence, even though the number of nodes of each tested graph may be increased, it will cost much more computing time.

Once finished, each experiment produces two data graphs, one describing the radius of the original covers and their coarsening covers, and one describing their degree; both as a function of the element examined in the specific experiment.

We define some important parameters for describing the experiment in practice:

- $sampled\ graphs$ = number of graphs tested in the experiment
- p = edge creation probability when generating a random graph
- $sampled\ covers$ = number of covers generated for each tested graph
- $cover\ size$ = a **minimum size** of a cover generated for the tested graph
- $\min\ \ cluster\ size$ = a **minimum size** of a cluster in a generated cluster cover
- $\max\ \ cluster\ size$ = a **maximum size** of a cluster in a generated cluster cover
- $K\ INTEGERS = [1, 4, 7, 10]$ is a list of k values sent to "MAX COVER" when calling it

Every experiment has its own adjusted parameters, with respect to its purpose and computing time. We will now discuss each of them separately:

1. Graph Density Experiment:

   For each edge creation probability $p \in$
   $[0.01, 0.06, 0.11, 0.16, 0.21, 0.26, 0.31, 0.36, 0.41, 0.46, 0.51, 0.56, 0.61, 0.66, 0.71, 0.76, 0.81, 0.86, 0.91, 0.96]$
   We generate $\boldsymbol{sampled\ graphs = 5}$ random graphs.
   For each random graph, we create $\boldsymbol{sampled\ covers = 5}$ random covers in size
   $\boldsymbol{cover\ size = 20}$.

For each random cover, we call "MAX COVER" with k, for every $k \in K\ INTEGERS$.

We sum up the calculated radius and degree of the original and the coarsening covers for each p, and then averaging it by dividing the sum by:

$sampled\ graphs * sampled\ covers * |K\ INTEGERS|$

2. Cover Size Experiment:

   We first generate $\boldsymbol{sampled\ graphs} = \mathbf{5}$ random graphs, with $p = 0.5$.

   Then, for each random graph, we create one of each cover types among the following six types:

Graph type	$\boldsymbol{cover\ size}$	$\mathbf{min}\ \boldsymbol{cluster\ size}$	$\mathbf{max}\ \boldsymbol{cluster\ size}$
Smallest cover	5	30	
…	10	20	
…	20	10	30
…	30	5	20
…	40	5	15
Largest cover	50		10

   For each cover type, we call "MAX COVER" with k, for every $k \in K\ INTEGERS$.

   We sum up the calculated radius and degree of the original and the coarsening covers for each cover type, and then averaging it by dividing the sum by:

   $sampled\ graphs * |K\ INTEGERS|$

3. K Integer Experiment:

   Again, we first generate $\boldsymbol{sampled\ graphs} = \mathbf{5}$ random graphs, with $p = 0.5$.

   For each random graph, we create $\boldsymbol{sampled\ covers} = \mathbf{5}$ random covers in size $\boldsymbol{cover\ size} = \mathbf{20}$.

   For each random cover, we call "MAX COVER" with k, for every $\mathbf{1} \leq \boldsymbol{k} \leq \mathbf{40}$.

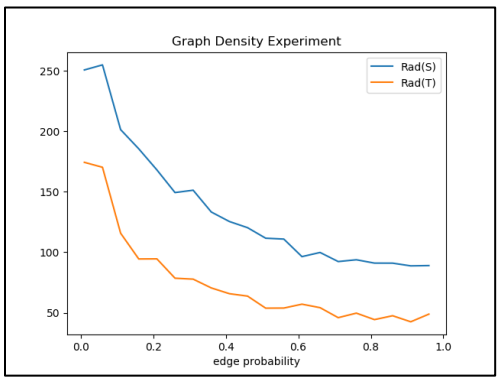
   We sum up the calculated radius and degree of the original and the coarsening covers for each k, and then averaging it by dividing the sum by:

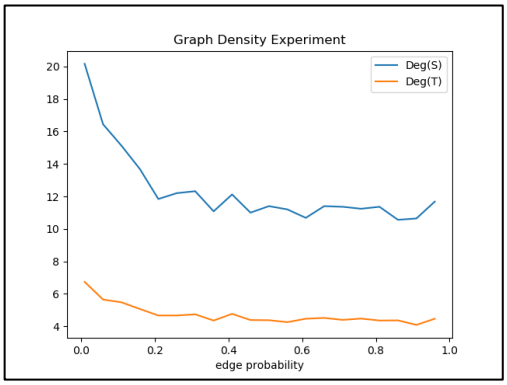
   $sampled\ graphs * sampled\ covers$

Experiments Results

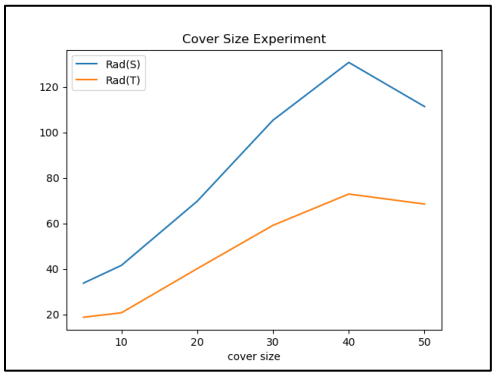
As mentioned above, every experiment produces two graphs. We present each experiment's result data graphs, where S denotes the original graph cover, and T denotes its coarsening cover, with respect to the algorithm's description in the article.

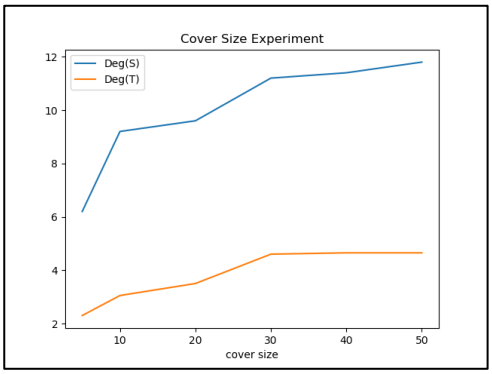
1. Graph Density Experiment:



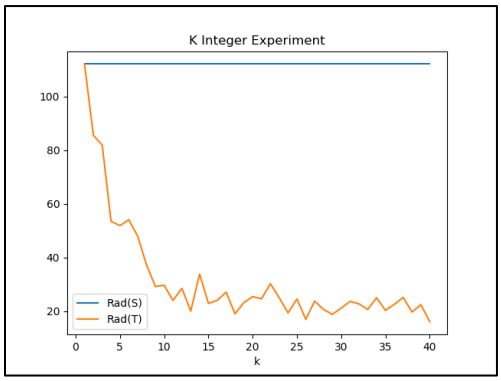


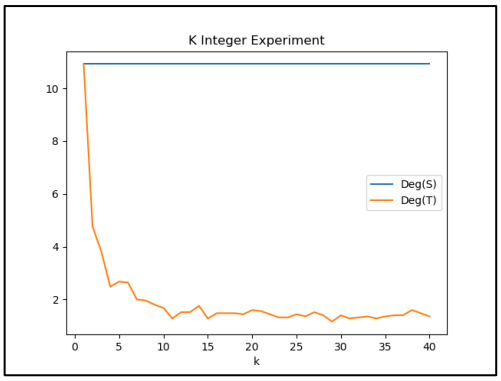
2. Cover Size Experiment:





3. K Integer Experiment:





Analysis and Conclusions

We first discuss each experiment on its own, and afterwards present general conclusions.

1. Graph Density Experiment:

   As we can see in the graphs, **the less dense (i.e. sparser) the graph is, the higher the radius and degree its clusters covers have**. At the same time, it is significant that the radius and degree of the coarsening covers act very similar, almost parallel to the original's, just with lower values. It is also noticeable that the change in covers radius is bigger than then change in their degree.

   We can find these results reasonable, since intuitively if the graph is sparse, then the harder it is to construct a clusters cover with "small size" clusters, or a cover with clusters that share high number of nodes. Hence, the covers have high radius and degree, when the graph is sparser.

2. Cover Size Experiment:

   Observing the graphs, it is apparent that **the larger the cover size is, the higher the radius and degree its clusters covers have**. Similarly to before, it is significant that the radius and degree of the coarsening covers act very similar, almost parallel to the original's, just with lower values. And again, it is noticeable that the change in covers radius is bigger than then change in their degree.

   We can explain these results by the logic that the larger the cover size is, the smaller its clusters are. Following that, the "size" of the clusters, as presented as a criteria for a qualitative cover, which means the distance between the nodes within each cluster, is relatively heavy weighted, since there aren't many paths between the nodes. This is why the radius of a large covers is big.

   In addition, because there a lot of clusters, more nodes are shared by different clusters, and therefore the degree of the cover is larger.

3. K Integer Experiment:

   As a reminder, the original cover's radius and degree don't depend on k, so their graphs are constant, representing the average radius and degree of the original covers in the experiment. Therefore, in this case we give more attention to the coarsening cover properties.

As we can see, the radius and the degree of the coarsening cover decrease very quickly, until they "converge" to some low value, in compare with the average original cover radius and degree.

Thus, we can conclude that the k parameter has great effect on the coarsening radius and degree, when **if k is large enough, the radius and degree will be relatively small**, again compared with the original cover degree.

General Conclusions:

Regarding the first two experiments that examine the effects of graph density and cover size, we found a lot of logic in the properties of the generated covers, with respect to the two basic principles for evaluating a qualitative cluster cover, Awerbuch and Peleg present in the paper.

An interesting finding in these experiments, is **the similarity between the data graph of the original cover (S) and its coarsening cover (T)**, concerning both radius and degree.

This means that **the algorithm provides coarsening covers, which have properties that are very proportional to the properties of the original covers, only with smaller values of radius and degree**.

In practice, this means that when looking for a qualitative representation for a network and we already have some clusters cover of it in our hands, if we run "MAX COVER" with this cover as input, we are somehow limited by the network sparsity and the original cover's quality. On the other hand, if we have the option to find a better cover, and then run the algorithm on it, we know it will likely pay off.

The first two experiments examine the effects of two elements, that are not necessarily changeable. However the third experiment, "K Integer Experiment", examines the effect of a parameter given to the algorithm by our own choice.

In this manner, we learned that **a large enough k might decrease dramatically the values of the coarsening cover radius and degree**. Practically, this means we get a more qualitative representation for a given network.

In an overall conclusion, we learn that the sparser the network, and the less qualitative its current representation of a clusters cover, the less qualitative the representation we get as a result from running "MAX COVER" on it (although we will get equal or better cover properties), and vice versa. Nonetheless, we can try to increase the k parameter sent to the algorithm, until a certain limit, and expect better results.

Future Work

The degree of a cover which represents its sparsity, is defined in this mini-project, and in accordance with “MAX COVER” output properties, in the following way:

$$Deg(S) = \max_{v \in V}\big(Deg(v, S)\big)$$

However, this definition is in fact of a cover’s **maximum degree**, which is one of two methods to describe a cover’s degree, presented in the article. The second way to define a cover’s degree is by its **average degree**:

$$Deg(S) = \frac{\sum_{v \in V} Deg(v, S)}{|V|}$$

Thus, it will be interesting to see the results of the experiments on “MAX COVER”, if we change the calculation of a cover’s degree to average degree.

Another intriguing direction to explore, which is also discussed in the paper, is the representation of a network as a clusters cover, that is also a **partition**. This means that every two clusters in the cover are disjoint. Obviously, using partition covers is better than using regular covers for networks representation, since partitions are absolutely sparse covers.

Following that, algorithms like “MAX COVER”, that might be able to coarsen a given partition cover will be very beneficial, since they will probably guarantee even better representation properties.