Network Community Detection in a Distributed Framework

Francisco Serrano
fjs@cpp.edu

Noah Reef
nbreef@cpp.edu

Mark Haddad
mahaddad@cpp.edu

Abanob Ayoub
akayoub@cpp.edu

Ryan Phan
ryanphan@cpp.edu

https://www.overleaf.com/project/6529fbd584df8dbfdd382432

***Abstract*—Within networked systems there could be hundreds of thousands of network communication within a single day, this makes tracking and tracing connections and the group they belong to difficult. Our goal is to use a dataset of 20 disjointed graphs across two days of data, each graph representing a connection among computers using Cisco's Secure Workload, to establish communities and clusters between the nodes/connections. We hope to see communities/clusters form between the nodes to see where nodes are forming groups and where in the groups the connections are coming from and going to. As we have progressed into extracting the most out of this dataset, we have established iterative performance measures to get a best model for forming these clusters.**

***Index Terms*—Community Detection, Networks, Clustering, Unsupervised, Cisco, Louvain**

I. Introduction

In networked systems, it is crucial to understand the structure and relationships found within complicated networks. This collaborative proposal represents a concerted effort to address these challenges by harnessing the power of distributed computing. By distributing the computational load across a network of interconnected nodes, we aim to design and implement algorithms that can efficiently detect communities in large-scale. Throughout this process we have understood that there are different approaches to solving this task. Our first focus was directed towards community detection, using Louvain's algorithm and the given ground truths. Over time we have shifted focus from community detection to graph analysis via clustering, where graphs are represented as vectors through a node embedding process, where we have then established an optimization of clustering these graphs through a parameter sweep. In this paper, we'll be applying clustering algorithms to a Cisco funded data set representing 20 distinct graphs. These graphs, sourced from real-world scenarios, provide a rich and diverse landscape for our research. By working with this comprehensive collection of graphs, each representing a unique network topology, we can rigorously test and optimize our clustering algorithms.

II. Data

For each graph of the entire dataset [1, Fig. 1], a graph can be represented by the connections of the nodes and its respective edges. A graph's features can further describe how the edges are connected, which ultimately are used to understand the communities within the network. Such features that can be extracted are for example, the direction of the edge from the consumer node to the provider node, the server port and network protocol (TCP and UDP), the number of directed $|E_d|$ and undirected $|E_u|$ edges within a graph, it's range of degrees for each of the nodes (d_m, d_{max}), as well as specification on the number of ports provided through an edge $(p_{1+}, p_{2+}, p_{max})$ and the number of packages exchanged. The graphs do not have any labeling, so the learning methodologies will be unsupervised, interpreted with various clustering algorithms. [1, Fig. 1] provides one graph that has ground truth grouping, which can be used for comparison as testing data. For our training parameter sweep and node embedding, we utilized CSV data files for each graph on each day, with the features being, the graph, source node, destination node, and a list of network protocols within that connection. From there we are able to extract weights from the packet size of the protocols, where different weight extractions can be used as part of the parameter sweep.

G	$\|V\|$	$\|E_u\|$	d_m	d_{max}	ports	ports2	$\|E_d\|$	p_{1+}	p_{2+}	p_m	p_{max}	c_{1+}	c_{2+}	c_m	c_{max}	p_1c_1
g4	161777	175536	1	141615	287	54	175610	740	43	1	196	161105	1071	1	226	68
g2	109634	1390120	2	19186	89168	69310	1597194	71258	24631	1	32376	78874	46779	2	18577	40498
g5	45876	54090	1	34853	2837	418	54658	44349	682	1	1359	2763	1320	1	1481	1236
g15	45706	71451	2	25782	1246	974	89295	13765	132	1	954	44313	3556	1	941	12372
g6	24032	91981	2	15009	4211	680	102153	15944	7330	1	281	19272	10234	6	3325	11184
g10	12614	27443	2	5958	5348	4259	27840	2019	988	1	4104	11047	1401	1	4129	452
g9	11683	23610	1	5736	1206	870	27180	5242	929	1	63	7585	2748	1	311	1144
g3	11214	42501	2	7645	11741	9611	48788	9402	2917	1	1968	8139	4603	2	8632	6327
g13	4517	16163	2	3025	1283	307	20206	2533	2055	3	47	4181	3280	8	337	2197
g8	3147	12382	5	2233	324	134	12440	406	99	1	119	2886	2221	9	64	145
g12	2145	3912	1	488	15331	10856	4369	1357	493	1	7615	1242	459	1	11424	454
g1	1263	97997	18	555	14857	13966	109554	908	601	6	697	779	628	4	6366	424
g17	525	830	1	247	107	47	871	128	37	1	62	446	144	1	55	49
g14	513	773	1	57	13355	201	822	168	50	1	12422	419	113	1	12422	74
g20	286	924	2	125	88	60	979	137	45	1	17	203	166	3	25	54
g18	270	465	2	216	34	19	467	43	15	1	9	232	196	3	20	5
g7	261	583	2	89	102	31	611	219	20	1	18	57	22	1	62	15
g16	214	275	1	107	61	44	283	102	27	1	26	124	69	2	23	12
g11	199	1522	7	98	61	38	1549	148	34	1	12	91	68	3	22	40
g19	79	133	1	44	32	24	133	66	16	1	10	18	7	1	21	5
g21	52	1282	50	51	2371	1398	2507	52	51	8	694	50	50	13	1161	50
g21'	52	682	21	51	2369	1396	952	52	51	6	692	50	50	11	1159	50

Fig. 1. The dataset comprised of 21 graphs. Each disjoint graph can represent one instance of the dataset, where each instance includes 16 features representing the network of connections within the graph. Furthermore, the features of the graph relate specifically to how the edges are connected

III. Our Work

In this study, we address the challenge of understanding complex network structures by employing distributed computing techniques to implement various clustering algorithms in Python for community detection across a Cisco-funded data set of 20 diverse real-world graphs. We emphasize data preparation, algorithm selection, and the use of distributed frameworks. Throughout the process, detailed documentation

will be maintained to ensure transparency and reproducibility. A version of the code used for our paper can be found at https://github.com/franserr99/communityDetection

A. Methodology

Following the community detection task outlined in [1], we implement an embedding-clustering algorithm to use graph data to embed nodes into a 2-dimensional space and utilize a clustering algorithm to find optimal partitions of our embedded graph data. As graphs are an unconventional form of data in a traditional machine learning pipeline, we have considered multiple ways of extracting different communities within the graphs. Machine learning techniques we've considered are Louvain's algorithm for community detection, Graph Neural Networks, supervised classification using the given ground truth graph, and unsupervised clustering. Going with the latter, a crucial component of data pre-processing to perform clustering is node embedding, and mapping a graph of nodes and edges in the form of "walks", to vectors that can be placed in a 2-dimensional space.

Graphs

For simplicity of implementation we separated each graph, G_j into a csv file by graph and day, $G_{j,day}$. We then computed new weights for each edge in a graph, $G_{i,j}$, by computing the number of communications between a source and destination node within a given time interval (usually a period of multiple hours).

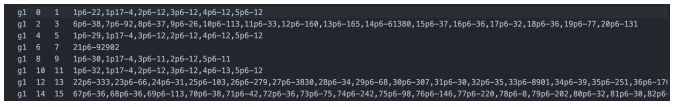

Fig. 2. $G_{1,1}$ without weight transformation

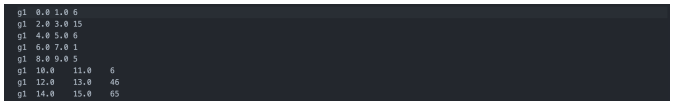

Fig. 3. $G_{1,1}$ with weight transformation

Weight Methods

We have various methods to calculate the weights of each edge in order to be used for the *Weighted Deepwalk*. These are total packet size, communication frequency, and normalized packet size. Normalize packet size was included as a method because of the extreme outliers within our data set that could heavily skew the strengths of the weights. These sub methods that we used to are averaging, log transformation, and a variation of Tukey's bi-weight mid-variance.

Total packet size is the summation of all the packets that were sent for that specific edge. Communication frequency refers to how many connections were made from that specific edge. Average weight was obtained by dividing the total packet size by the communication frequency. We had use log transformation in order to reduce the skewedness of our data by trying to make the distribution symmetric. Originally, we had thought of using z-score distribution in order to normalize the weights but that was proven to be a bad idea as z-score distribution takes account the mean and standard deviation. We were using data sets that had packet sizes in the millions, using z-score distribution would heavily impact the normalize weights, so we switched to a variation of Tukey's bi-weight mid-variance. This variation allows for us to use our data sets properly because the outliers within our data (the packets size in millions) would have less of an impact within our normalized weights. The driving factor behind this method is the use of the packet median (rather than the mean). Below is the formula of this variation that we used. The normalized weight for that edge is the edge's average subtracted by the median and then all of that is divided by a tuning factor. The tuning factor is the denominator and in which can be change in any shape depending on how strong we want to tune the weights. C is just a constant number.

$$Weight(Edge) = \frac{Average(Edge) - Median}{C * Median}$$

Graph Embedding

Next we embed each graph by using the *Deepwalk* implementation in the `Karateclub` library to embed our graph into a two-dimensional plane

Algorithm 1 DeepWalk(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t
Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w)
  end for
end for

Fig. 4. Deepwalk Algorithm [2]

this gives us a 2-dimensional mapping of the data that we can use for future attempts of clustering algorithms. For our initial parameter sweep we ran both `Stellargraph`'s [3] *Weighted Deepwalk* and *Unweighted Deepwalk* to find the best embedding technique.

Clustering

After performing embedding on our graph data, we look to implement a clustering algorithm to find an optimal partition of our embedded data which correspond to the found communities of our model. We perform a parameter sweep over the following clustering algorithms: *K-Medidiods, K-Means,*

Spectral Clustering, and Density-Based Spatial Clustering of Applications with Noise (DBSCAN). We then evaluate the performance of each clustering algorithm using the *Silhouette Score*.

RESULTS

After running a parameter sweep over the methods described above using

$$S = \{G_{1,1}, G_{13,1}, G_{12,1}, G_{20,1}, G_{17,1}, \dots \\ G_{14,1}, G_{7,1}, G_{18,1}, G_{16,1}, G_{11,1}, G_{19,1}, \}$$

as our training set, we found our best model to be with an embedding technique of *Unweighted Deepwalk* and *DBSCAN* as our clustering technique.

The silhouette score for a subset of our testing set is shown below.

graph	silhouette
$G_{18,1}$	0.5995639
$G_{1,2}$	0.58565176
$G_{20,3}$	0.5232006
$G_{2,3}$	0.6734304
$G_{7,1}$	0.5942679

and we found that the average silhouette score was 0.607006.

CONCLUSION

Following the aims of work discussed in [1], we implemented a embedding-clustering approach, after considering a parameter sweep of both different clustering and embedding techniques, utilizing *DBSCAN* as our clustering technique and *Unweighted Deepwalk* as our embedding technique to perform *community detection*. Using the model described above and *silhouette score* as our performance metric, we found that our model had an average score 0.607006 in cluster quality. From this measure, we can conclude that the graphs contain reasonably detectable communities that are different from one another. Because of our fully unsupervised technique, and the nature of the network graphs, it is still unknown exactly how or what factors differentiate the communities. Nonetheless, we know they are differentiable from our results, extracting a good partition of clusters.

FUTURE WORK

As stated earlier in our *results* section, we found that the optimal embedding technique was that of *Unweighted Deepwalk* which ignores graphs weights in determining the embedding of the graph. It may be possible that the out performance of *Unweighted Deepwalk* against *Weighted Deepwalk* may be possibly due to how accurate the weight characterization reflects the connection between nodes in the graph. As stated in our *methodology* section we characterized the weights of our network graphs by the number of communications between a source and destination node over a period of time. For future work, we would like to investigate other characterization of the weight between two nodes in our graph by using different metrics in classifying weights such as number of distinct ports or total packets sent as a better proxy than the number of communications. This change of classification in weights may also have an impact on which embedding technique will have better performance.

Another area of future interest is in identifying and classifying community roles in the network. The model described in our paper does not provide any additional insight about the relationships between nodes in a given cluster nor any additional data relating to their role in their respective service, but instead tries to provide an optimal partitioning of the embedded data. Developing extensions to detect roles within a community may be additional area worth investigating.

SUPPLEMENTARY MATERIAL

Paper:
https://www.overleaf.com/project/6529fbd584df8dbfdd382432
Source-Code:
https://github.com/franserr99/communityDetection

REFERENCES

[1] "A Dataset of Networks of Computing Hosts", IWSPA 2022. Omid Madani, Sai Ankith Averineni, Shashidhar Gandham, IWSPA 2022.

[2] "DeepWalk: Online Learning of Social Representations", KDD 2014. Bryan Perozzi, Rami Al-Rfou, Steven Skiena.

[3] StellarGraph, CSIRO's Data61, StellarGraph Machine Learning Library,https://github.com/stellargraph/stellargraph