Review of Community Detection in Complex Networks Using Cellular Learning Automata

Sy Fontenot

***Abstract*—In this report, we review a paper by Zhao et al. [1] who proposed a unique approach to community detection in complex networks by implementing the theory of cellular learning automata. We further detail aspects of the algorithm that the authors of the paper omit in lieu of explaining the broader ideas. We then compare the results of some of the real-world networks.**

I. Introduction

COMPLEX networks are mathematical structures used in modeling many real-world systems such as social communications, router traffic and connectivity, hyperlinks between web pages, biochemical interactions, and many others. The analysis of complex networks as a whole has thus become increasingly important in recent literature. Due to the many applications in which networks can be utilized to model such systems, various metrics such as centrality, community, and growth rates have been developed, each potentially having vastly different underlying rationale. Describing a single means of centrality measure may do well for predicting importance for one network, but it may be next to meaningless when implemented onto another network. Along with the network being studied, it is also dependent on the questions being asked of such network. Degree centrality, for example, may be a good indicator in a social network to evaluate from which individuals information might best permeate, but it does not work well when determining traffic buildup in internet routers.

Communities prove to be another useful measure when studying complex networks and is the main topic of concern in this report. Along with centrality, the subfield of community detection has produced countless algorithms with varying proficiencies and shortcomings. Some may perform well in regards to execution time and detecting communities of large networks but waver in power-law distribution benchmarks [2] Others may appear lethargic, performing moderately on random benchmarks but resulting in highly accurate and precise measures on most standard real-world networks [3].

In this report, we will be following an algorithm that claims to excel in both random and real-world network benchmarks—displaying effectiveness in both speed and performance. This algorithm by Zhao et al. implement their unique solution with cellular learning automata to model the network structure [1]. The individuality of these cellular automata introduces and complex emergent behavior that allows for the algorithm to solve resolution limit issues that are fundamental to many global implementations of community detection algorithms [4]. In an attempt to deviate from the original paper itself which outlines the background, ideas, and results of the algorithm, we will instead delve more into the details of its execution that have been glossed over in the paper. We will then compare the results we produce with that of the original experiments.

II. Background

Before continuing to the methodology, however, we still believe it beneficial to give a brief summary of the concepts leading up to the algorithm.

Measures

Outlining the famous metric by Newman and Girvan called modularity [5]. Modularity soon became the main mechanism beneath many of the best community detection algorithms. That is, the entire problem of community detection is often attributed to the maximal optimization of this single metric. The following definition of modularity Q was used:

$$Q = \frac{1}{D} \sum_{i,j \in V} \left(A_{ij} - \frac{d_i d_j}{D} \right) \delta(i,j) \tag{1}$$

where D represents the degree count, V represents the set of vertices, A represents the adjacency matrix of the network, d_i represents the degree of node i, and δ represents the community relationship between nodes i and j such that $\delta(i,j) = 1$ if nodes i and j exist in the same community, and $\delta(i,j) = 0$ otherwise.

From here, two researchers by the names Fortunato and Barthélemy proposed a shortcoming with modularity optimization they called the resolution limit [4]. They found that the detection of communities becomes obsolete below a certain size. As detailed in their paper, they claim that modularity as a single metric is not often sufficient enough when searching for its own optimization.

It is also important to define community structure itself. One definition used for determining community structure, as proposed by Raghavan et al. [6], is written as

$$k_i(c) \geq k_i(c'), \quad \forall i \in c, \quad \forall c' \in \Omega \tag{2}$$

where $k_i(c)$ is the number of neighboring nodes in community c connected to node i, and Ω is the partition of the network into communities.

These are standard metrics utilized by algorithms aimed to find communities of a network. Therefore, where each algorithm might largely differ would be in how they permute through the space of all possible community partitions of the network. This proves increasingly difficult as the size of the network gets large.

Cellular Learning Automata

During our exploration of the literature, one notable approach to community detection by Zhao et al. in 2014 [1] captivated our attention, and its replication soon became the focus our project. Zhao et al. proposed an algorithm they called CLA-net which utilizes the theory of cellular learning automata to better localize movement through the search space. They claimed to resolve Fortunato's and Barthélemy's resolution limit, having found better results than several of the leading algorithms of the time.

As a brief summary, cellular automata are a set of identical cells arranged in a regular lattice that each partake in localized interactions [7]. Each cell belongs to a finite set of states, and each cell state may change according to a local rule. Cellular automata have been widely studied in many different subjects of literature, but its use in network community detection is, as far as we have seen, rather novel.

A learning automaton is a model similar to the standard automaton of a cell previously described. However, instead of simply having a set of states to change to (called the action set), learning automata have an additional action probability vector that stores the probability of selecting a given action (or state) in the next iteration [8]. The local rule for a learning automaton thus alters this probability vector rather than it choosing the action deterministically. The action is then chosen based on the local probability distribution of the automaton. The automata is said to be learning as the probability vector changes with respect to feedback from the environment.

Cellular learning automata (CLA) simply combine the previous ideas of cellular and learning automata. That is, each cell contains a learning automaton. To apply this to the study of complex networks, the regularity of a lattice must be relaxed. Thus, irregular cellular learning automata refer to cellular learning automata that operate without the constraints of a regular lattice. Furthermore, the system is called open if the automata learn from a global environment as well as locally, and the system is synchronous if the evolution of each cell occurs at the same time. The problem of community detection suggests the usage of open synchronous cellular learning automata (OSCLA) [9]. Note that, despite the name, this CLA is irregular.

Representation

Lastly, the problem of community detection largely falls ignorant to the optimal number of communities. Therefore, a means to store community structure must remain independent of an assumed fixed quantity. Ideally, community structure is represented as a membership vector $C = (c_1, ..., c_n)$ where c_i is the index of the community that node i belongs to. Instead, it is best to implement a solution vector $S = (s_1, ..., s_n)$ where s_i indicates that nodes i and s_i belong to the same community [10]. This solution vector can then be decoded into the desired membership vector.

We can now discuss our implementation of the CLA-net algorithm. As previously mentioned, we will be highlighting the aspects of the network that the paper omitted, leaving the rest as encouragement for the reader to review the original paper themselves.

III. Methodology

CLA-net utilizes both the local and the global environment when determining the probability vectors of its learning automata. It thus employs open synchronous cellular learning automata. From the global environment, the modularity from Eq. (1) is calculated to determine how well the algorithm has partitioned the network. From the local environment, each learning automaton records the communities of its neighbors. If the learning automaton both improves the modularity and satisfies the Raghavan definition of Eq. (2), then it is rewarded. Otherwise, it is penalized. This determines how each learning automaton evolves.

Input

The algorithm begins by inputting the adjacency matrix $A_{n\times n}$ of the network $G = (V, E)$ where V is the set of nodes, E is the set of edge connections between the nodes, and $n = |V|$ is the node count. Each run of the algorithm has with it a fixed reward parameter $\alpha \in (0, 1)$ that is used in modifying the probability vectors of each learning automaton.

Data Structures

Four other structures are used in storing the data of the learning automata:

- `actionCountArray`: a vector r denoting the number of actions (or the degree) of each node in the network. The ith element of this vector r_i is calculated by summing up the elements of ith column or row.
- `actionRewardCountMatrix`: a matrix $W_{n\times n}$ denoting the number of times an action has been reward for each learning automaton. Each time an action from node j corresponding to node i is taken and rewarded, W_{ij} is incremented.
- `actionChosenCountMatrix`: a matrix $Z_{n\times n}$ denoting the number of times an action has been taken for each learning automaton. Similar to W, each time an action from node j corresponding to node i is taken, Z_{ij} is incremented.
- `actionProbabilityMatrix`: a matrix $P_{n\times n}$ denoting the probability vectors of each learning automaton. That is, P_{ij} represents the probability of node j selecting action i. Note that each column vector sums to 1.

Additionally, the best modularity Q_{best} is stored.

Functions

A set of five helper functions have been written before reproducing the algorithm to better modularize the code.

- `findModularity`: This function returns the result Q of the equation for modularity Eq. (1). Both a membership vector C and adjacency matrix $A_{n\times n}$ are inputs. In the original paper, the degree count D is represented as $2m$ where m is the edge count of the network. In practice,

however, it is easier to simply sum up the elements of the adjacency matrix in a single `numpy.matrix` function.

- `solutionToMembershipVector`: This function decodes an inputted solution vector $S = (s_1, ..., s_n)$ into its corresponding membership vector $C = (c_1, ..., c_n)$ and returns the results. To achieve this, a vector of size n is filled such that each element is the number n itself. A community counter starts at 0. For each node i, if $c_i \neq c_{s_i}$, then the function enters a `while` loop with this condition which updates the values c_i and c_{s_i} to $\min(c_i, c_{s_i})$ and sets the new index to be s_i. The effect of this is to connect the nodes together whose actions point to each other. When the `while` loop terminates, the most recent index m is used to check if c_{s_m} is equal to n. This occurs when the last node in that particular sequence of connected nodes points to an action node that has not yet been identified with a community (it has default value n). This unclaimed node gets assigned the value of the community counter which then increments by 1. Iterating through each node once, however, often only links nodes into subcomminuties of their intended community structure. Thus, this is repeated n times to ensure the properly encoded structure.
- `chooseActionVector`: This function returns the action vector S (this acts as the solution vector) after inputting the action probability matrix $P_{n\times n}$. For each node j, the jth column of P is taken as a probability distribution. That is, node j has a probability of P_{ij} to select action node i. Note that if nodes i and j do not connect, then $P_{ij} = 0$.
- `getCommunityStructure`: This function has any membership vector C as input and returns list of lists L where each inner list is the set of nodes in a community. We find the set of distinct communities indices in C. For each distinct community index i, a list of all indices of C whose elements equal i are appended to the initially empty list L.
- `satisfiesRaghavan`: This function returns the evaluted Raghavan's condition as presented in Eq. (2). As inputs, there is the node i, the membership vector C and the adjacency matrix $A_{N\times n}$. From C can be found the community structure Ω as a list of mutually exclusive lists of nodes. The function iterates through each community $c \in \Omega$. The set of nodes within the same community as node i are determined by $c_{in} = c$ if $i \in c$. If $i \notin c$, then the nodes in c are appended to the list c_{out}. The function returns true if and only if the number of nodes in c and connected to node i is greater than or equal to the number of nodes connected to node i and not in c.

Although the `solutionToMembershipVector` function works, it is subject for improvement. This may cause unnecessary overhead that hinders timely execution of the program for large networks.

Algorithm

In Step 1 of the CLA-net algorithm, we initialize each column vector of P to produce a uniform distribution. Note that when iterating through the matrix, we must first check if the corresponding action vector element r_i is not zero before taking its reciprocal. That is, the node i must connect to at least one node.

In Step 2, an action vector is chosen based on the action probability matrix by `chooseActionVector`, initializing values in W, Z, and Q_{best}. That is, for each node i, the chosen action matrix Z at index (i, s_i) is incremented where s_i is node i's chosen action. If the calculated modularity is greater than or equal Q_{best} and Raghavan's condition is met, then Q_{best} is updated and W at index (i, s_i) is incremented. The paper suggests to do this "a small number of times", so we have chosen to execute this portion 5 times. It is currently unclear as to whether or not different number of repetitions will persuade converges too little, too much, or not at all.

The rest of the algorithm exists within a `while` loop until the community structure remains fixed after consecutive iterations. However, our implementation simply runs for 10000 iterations before returning. This is to be properly applied in the future.

Steps 3-8 are simply a repeat of Step 2 but are now affected by the change in probability from Step 10.

Step 9 finds the optimal action vector a which is described as the highest action probability for each node. For each node j, a_j is the node such that $P_{ij} = \max(P_{1j}, ..., P_{nj})$ where P is the action probability matrix and n is the number of nodes.

Finally, Step 10 updates the action probability matrix P using the previous calculated optimal action vector a according to the following equation. For each node j,

$$P_{ij}(t+1) = \begin{cases} P_{ij}(t) + \alpha(1 - P_{ij}(t)) & j = a_j \\ (1-\alpha)P_{ij}(t) & j \neq a_j \end{cases} \tag{3}$$

where $P_{ij}(t)$ represents P_{ij} at iteration t and α is the algorithm's fixed reward parameter.

The original CLA-net algorithm outputs the solution vector S, membership vector C, and the corresponding community structure. Since our attempt is yet to terminate as intended, the action probability matrix does not converge, so neither does the optimal action vector. Therefore, we output the best modularity found by the algorithm along with its membership vector.

IV. Experiments

In the original paper, Zhao et al. test their algorithm on both synthetic and real-world networks, comparing their results to competing community detection algorithms. For our purposes, we will only be finding results for the real-world networks—viz. Zachary's Karate club [11] with 34 nodes and 78 edges, Lusseau's dolphins [12] with 62 nodes and 159 edges, and American College football union [13] with 115 nodes and 616 edges. The other real-world networks used in the paper are either unavailable, too large for our non-optimized code, or have single-node components (our implementation is yet to handle such networks). Since this project is a review of the CLA-net algorithm, we will simply compare our results to that of the paper.

TABLE I
CLA-NET: ORIGINAL RESULTS VS OUR RESULTS

Network		Original	Ours	Average Time (s)
Karate	Q_{max}	0.4188	**0.4198**	22.8
	Q_{avg}	0.4175	**0.4192**	
Dolphin	Q_{max}	0.5277	**0.5285**	67.9
	Q_{avg}	**0.5268**	0.5244	
Football	Q_{max}	**0.6046**	0.6032	226.9
	Q_{avg}	**0.6042**	0.5923	

By trial and with little guidance from the paper, we settled on a reward parameter of $\alpha = 0.0005$. We initially set the parameter to 0.5, but the results converged too quickly to an solution that was far from optimal. Only after revisiting Eq. (3) did we understand that this value should be near 0 for larger networks and thus larger search spaces.

As mentioned, we have not yet introduced the proposed termination condition for Steps 3-10. Thus, we have chosen to cycle through 10000 times before the algorithm returns the best modularity found. By doing this 10 times for each network, we have been able to find a maximum and average modularity measure as presented in Table I.

V. RESULTS

As seen in Table I, our results rival those of the original paper. At times, our runs even outperformed those of the paper. It is interesting to note, also, that our maximum run of the karate network tied that of the best algorithm's run in the original paper.

This success, however, is only in metric, as the time it takes to execute the algorithm is far longer than that suggested by the results of Zhao et al.

VI. CONCLUSION

In this report, we have demonstrated the implementation of the CLA-net algorithm that uses irregular cellular learning automata in solving the problem of community detection. We have shown that such replication is both possible and may lead to better results. We have detailed the elements of the algorithm that have not been thoroughly explained. As a first outcome of this report, we hope that our description may better help the reader understand the CLA-net algorithm in its finer resolution. Secondly, this report acts as a foothold to move forward by, as there are many enhancements to be had in optimizing our code and finishing out the rest of the proposed features.

Once these objectives have been met, future study of this project will be in researching potential improvements to the ideas at large—possibly combining the cellular learning automata with other data structures or ideas.

REFERENCES

[1] Y. Zhao, W. Jiang, S. Li, Y. Ma, G. Su, and X. Lin, "A cellular learning automata based algorithm for detecting community structure in complex networks," *Neurocomputing*, vol. 151, pp. 1216–1226, 2015.

[2] F. Hu and Y. Liu, "A new algorithm cnm-centrality of detecting communities based on node centrality," *Physica A: Statistical Mechanics and its Applications*, vol. 446, pp. 138–151, 2016.

[3] C. Shi, Z. Yan, Y. Cai, and B. Wu, "Multi-objective community detection in complex networks," *Applied Soft Computing*, vol. 12, no. 2, pp. 850–859, 2012.

[4] S. Fortunato and M. Barthelemy, "Resolution limit in community detection," *Proceedings of the national academy of sciences*, vol. 104, no. 1, pp. 36–41, 2007.

[5] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E*, vol. 69, no. 2, p. 026113, 2004.

[6] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical review E*, vol. 76, no. 3, p. 036106, 2007.

[7] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of modern physics*, vol. 55, no. 3, p. 601, 1983.

[8] M. A. Thathachar and P. S. Sastry, "Varieties of learning automata: an overview," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 32, no. 6, pp. 711–722, 2002.

[9] H. Beigy and M. R. Meybodi, "Open synchronous cellular learning automata," *Advances in Complex Systems*, vol. 10, no. 04, pp. 527–556, 2007.

[10] C. Pizzuti, "Ga-net: A genetic algorithm for community detection in social networks," in *International conference on parallel problem solving from nature*. Springer, 2008, pp. 1081–1090.

[11] W. W. Zachary, "An information flow model for conflict and fission in small groups, 1977," *Journal of Anthropological Research B (4)*, pp. 452–473.

[12] D. Lusseau, "The emergent properties of a dolphin social network. eprint," *arXiv preprint cond-mat/0307439*, 2003.

[13] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.