A Dynamic Modularity Based Community Detection Algorithm for Large-scale Networks: DSLM

Riza Aktunc and Ismail Hakki Toroslu
Computer Engineering Department
Middle East Technical University
Ankara, Turkey 06530
Email: {riza.aktunc, toroslu}@ceng.metu.edu.tr

Mert Ozer and Hasan Davulcu
School of Computing, Informatics,
Decision Systems Engineering
Arizona State University
Tempe, USA 85287
Email: {mozer, hdavulcu}@asu.edu

***Abstract*—In this work, a new fast dynamic community detection algorithm for large scale networks is presented. Most of the previous community detection algorithms are designed for static networks. However, large scale social networks are dynamic and evolve frequently over time. To quickly detect communities in dynamic large scale networks, we proposed dynamic modularity optimizer framework (DMO) that is constructed by modifying well-known static modularity based community detection algorithm. The proposed framework is tested using several different datasets. According to our results, community detection algorithms in the proposed framework perform better than static algorithms when large scale dynamic networks are considered.**

I. Introduction

In the last decade, the notion of social networking is emerged and produced very large graphs that consist of the information of their users. These graphs generally consist of the nodes that represent the users; and edges that represent the relations among users. The nodes in these graphs generally tend to get together and construct communities of their own. Thus, it can be stated that social networks commonly have a community structure. These networks can be divided into groups of nodes that have denser connections inside the group; but fewer connections to the outside of the group. For example, in a GSM network, a group of users who call each other more densely than they call other users may construct their own community. In this case, the nodes represent the users and the edges represent the calls that users made. The detection of communities in these large networks is a problem in this area; therefore a lot of community detection algorithms such as [1], [2], [3], [4], [5], [6], [7], [8], [9] proposed in the literature. Almost all these community detection algorithms are static and designed for static networks.

However, most of the social networks are not static because they evolve in many ways. They may gain or lose users that are represented as nodes in the graphs over time. The users of these social networks may lose contact from each other or there can be new connections among users. In other words, some edges in the graphs may be removed or new edges may be added to the graph over time. All these processes may happen in a very small amount of time in a social network if it has a lot of active users. This kind of a social network may be called as highly dynamic. For example, popular social sites such as Facebook, Twitter, LinkedIn and so on have highly dynamic social networks. Moreover, most GSM networks have millions of users and hundreds of calls made in seconds; therefore, they can also be labeled as highly dynamic networks. Addition or deletion of an edge or a node from a network which has millions of edges might seem insignificant; but when this additions or deletions of an edge or a node happen very frequently, they begin to change the community structure of the whole network and become very important. This change in the community structure raises the need of re-identification of communities in the network. This need arises frequently and creates a new problem in the community detection research area. This new problem requires somehow fast detection of communities in dynamic networks.

The first solution that comes to mind for community detection in large dynamic networks problem is the execution of static community detection algorithms already defined in the literature all over again to detect the new community structure whenever the network is modified. Nevertheless, this solution takes too much time in every modification of the large networks since it runs the community detection algorithm from scratch each time. A much efficient and less time consuming solution is to run the community detection algorithms not from scratch but from a point in the history of the network by storing and using the historical results of executions of the algorithms whenever network is evolved. In other words, updating previously discovered community structure instead of trying to find communities from scratch each time the network evolves consumes much less time and thus much efficient. This solution method for the problem of detecting communities in large dynamic networks is the main focus of our study in this paper.

In this paper, we modified the smart local moving (SLM) algorithm defined by Waltman & Van Eck [9] so that it would detect the communities in rapidly growing large networks dynamically and efficiently. As a result, we propose the dynamic SLM (dSLM) algorithm that dynamically detects communities in large networks by optimizing modularity and using its own historical results. We tested our proposed approach on several different datasets. We demonstrated the effects of our contribution to the SLM algorithm in two ways. One of them is the change in modularity value which determines the quality of the community structure of the network. The other one is the change in running time that determines the pace of the algorithm. The latter is more significant than the former because the community structure of the network must be quickly identified at the given timestamp before the next

timestamp is reached. We realized that dSLM improved SLM by decreasing its running time incredibly. Moreover, there are some experiments where modularity value increases while running time decreases.

The rest of the paper is organized as follows. Section II introduces previous researches done in the area. Section III explains the modularity. The proposed solution for dynamic community detection in large networks called as dSLM and its static version SLM are described in Section IV. In Section V, the results of the experiments of SLM and dSLM are demonstrated. Finally, the paper is concluded in Section VI.

II. Related Work

The idea of modularity-based community detection is to try to assign each vertex of the given network to a community such that it maximizes the modularity value of the network. Optimizing modularity is an NP-hard problem. [10] Exact algorithms that maximize modularity such as [11], [10], [12] can be used only for small networks.

For large-scale modularity optimization, heuristic algorithms are proposed. We basically focus on three well known algorithms, namely; CNM, Louvain and SLM. The first one is Clauset et al.'s [13] CNM algorithm. It is a greedy modularity maximization algorithm that searches for best community assignment for each node. The second one is referred as Louvain algorithm and proposed by Blondel et al. [7] in 2008. By considering each community as a single node, it further searches for new community merges after the local optimum satisfied using CNM. The last one is called as Smart Local Moving (SLM) algorithm that is proposed by Waltman and Jan van Eck in 2013. [9] SLM algorithm is explained in detail in chapter III.

Due to the dynamic features of many social networks [14], the need for detecting communities dynamically in the large networks is emerged in the latest years. There have been many community detection algorithms proposed in the literature to fulfill this need. Xu et al. divides the current research on community evolution into the following categories. Parameter estimation methods and probabilistic models have been proposed in the literature. [15], [16] A methodology that tries to find an optimal cluster sequence by detecting a cluster structure at each timestamp that optimizes the incremental quality can be classified as evolutionary clustering. [17], [18] Furthermore, tracking algorithms based on similarity comparison have also been studied in order to be able to describe the change of communities on the time axis. [19], [20] Apart from these algorithms that are focused on the evolution procedures of communities, community detection in dynamic social networks aims to detect the optimal community structure at each timestamp. For this purpose, incremental versions of both CNM and Louvain algorithm are proposed by Dinh et al.[21] and Aynaud et al. [22]. To the best of our knowledge, this is the first work considering the incremental version of Smart Local Moving algorithm in literature. Our algorithm can be classified as the last mentioned category which aims to detect optimal community structure at each timestamp with minimum running time.

III. Modularity

Modularity is a function that is used for measuring the quality of the results of community detection algorithms. If the modularity value of a partitioned network is high, it means that the network is partitioned well. Apart from quality measurement, modularity is used as the basis of some community detection algorithms. These algorithms try to detect communities (partitions) in a network by trying to maximize the modularity value of the network. Thus, modularity is a function that is used for both quality measurement and community detection.

Modularity is based on the idea that a randomly created graph is not expected to have community structure, so comparing the graph at hand with a randomly created graph would reveal the possible community structures in the graph at hand. This comparison is done through comparing the actual density of edges in a subgraph and the expected edge density in the subgraph if the edges in the subgraph were created randomly. This expected edge density depends on how random the edges created. This dependency is tied to a rule that defines how to create the randomness and called as null model. A null model is a copy of an original graph and it keeps some of this original graphs structural properties but not reflects its community structure. There can be multiple null models for a graph such that each of them keeps different structural properties of the original graph. Using different null models for the calculation of the modularity leads to different modularity calculation methods and values. The most common null model that is used for modularity calculation is the one that preserves the degree of each vertex of the original graph. With this null model, modularity is calculated as the fraction of edges that fall in the given communities minus such fraction in the null model. [23], [24] The formula of modularity can be written as in Equation 1

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j) \tag{1}$$

m represents the total number of edges of the graph. Sum iterates over all vertices denoted as i and j. A_{ij} is the number of edges between vertex i and vertex j in the original graph. P_{ij} is the expected number of edges between vertex i and vertex j in the null model. The δ function results as 1 if the vertex i and vertex j are in the same community ($C_i = C_j$), 0 otherwise. The null model can be created by cutting the edges between vertices; thus, creating stubs (half edges) and rewiring them to random vertices. Thus, it obeys the rule of keeping degrees of vertices unchanged. Cutting edges into half, creates $m * 2 = 2m$ stubs. In the null model, a vertex could be attached to any other vertex of the graph and the probability that vertices i and j, with degrees k_i and k_j , are connected, can be calculated. The probability p_i to pick a ramdom stub connection for vertex i is $\frac{k_i}{2m}$, as there are k_i stubs of i out of a total of $2m$ stubs. The probability of vertex i and vertex j being connected is $p_i p_j$, since stubs are connected independently of each other. Since there are $2m$ stubs, there are $2m p_i p_j$ expected number of edges between vertex i and vertex j. [24] This yields to equation 2

$$P_{ij} = 2mp_ip_j = 2m\frac{k_ik_j}{4m^2} = \frac{k_ik_j}{2m} \quad (2)$$

By placing equation 2 into equation 1, modularity function is presented as in equation 3.

$$Q = \frac{1}{2m}\sum_{ij}(A_{ij} - \frac{k_ik_j}{2m})\delta(C_i, C_j) \quad (3)$$

The resulting values of this modularity function lie in the range $\left[\frac{-1}{2}, 1\right)$. It would be positive if the number of edges within subgraphs is more than the number of expected edges in the subgraphs of null model. Higher values of the modularity function mean better community structures. [9]

This modularity function also applies to weighted networks. [9], [25] The modularity function for the weighted graphs can be calculated as in equation 4.

$$Q_w = \frac{1}{2W}\sum_{ij}(W_{ij} - \frac{s_is_j}{2W})\delta(C_i, C_j) \quad (4)$$

There are three differences. The first difference is that in the case of a weighted network W_{ij}, instead of A_{ij}, may take not just 0 or 1 but any non-negative value that represents the weight of the edge. The second one is that instead of m, which is total number of edges, W, which is the sum of the weights of all edges is used in the equation. The last one is that s_i and s_j which represents the sum of the weights of edges adjacent to vertex i and vertex j respectively is used in the equation instead of k_i and k_j which means the degree of vertex i and vertex j respectively. [24]

Apart from weighted networks, the modularity function defined in 3 has been extended in order to be also applicable to directed networks. [26], [27] When the edges are directed, stubs will also be directed and it changes the possibility of rewiring stubs and connecting edges. The calculation of this possibility in the directed case depends on the in- and out-degrees of the end vertices. For instance, there are two vertices A and B. A has a high in-degree and low out-degree. B has a low in-degree and high out-degree. Thus, in the null model of modularity, an edge will be much more likely to point from B to A than from A to B. [24] Therefore, the expression of modularity for directed graphs can be written as in equation 5

$$Q_d = \frac{1}{m}\sum_{ij}(A_{ij} - \frac{k_i^{out}k_j^{in}}{m})\delta(C_i, C_j) \quad (5)$$

The sum of the in-degrees (out-degrees) equals m not $2m$ as in the case of undirected graph. Therefore, the factor 2 in the denominator of the first and second summand has been dropped. In order to get the modularity function to be applicable to directed weighted networks, the equations 4 and 5 can be merged; thus, equation 6 can be constructed as the most general expression of modularity. [24]

$$Q_{dw} = \frac{1}{W}\sum_{ij}(W_{ij} - \frac{s_i^{out}s_j^{in}}{W})\delta(C_i, C_j) \quad (6)$$

There have been a few proposals of modified version of the modularity functions defined above as alternative modularity functions. These modified, extended versions for instance offer a resolution parameter that makes it possible to customize the granularity level at which communities are detected and to mitigate the resolution limit problem defined by Fortunato and Barthlemy [28]. [29] Moreover, there are modularity functions with a somewhat modified mathematical structure in the literature such as Reichardt & Bornholdt, 2006; Traag, Van Dooren, & Nesterov, 2011; Waltman, Van Eck, & Noyons, 2010. [9], [28], [30], [31]

IV. SLM and dSLM Algorithms

A. SLM Algorithm

SLM is a community detection algorithm that is evolved from Louvain algorithm. Louvain algorithm is a large scale modularity based community detection algorithm that is proposed by Blondel et al in 2008. [7] The quality of detected communities by Louvain algorithm is measured by the method called modularity. The modularity of a network is a value that is between -1 and 1. This value presents the density of links inside communities over the density of links between communities. [23] When this value is close to 1, then the measured network can be called as modular network. In the case of weighted networks, modularity function can take weights into consideration and measure the quality of detected communities. Louvain algorithm uses modularity function as not only a measurement function but also an objective function to optimize.

Louvain algorithm is a recursive algorithm which has two steps running in each recursive call. Before the recursion starts, the algorithm assigns a different community to each node of the network whose communities are going to be detected. Therefore, in the initial case each node has its own community. In each recursive call the following steps are run:

1) It runs a local moving heuristic in order to obtain an improved community structure. This heuristic basically moves each node from its own community to its neighbors' community and run the modularity function. If the result of the modularity function, which means quality, increased, the node would be kept in the new community; else, the node would be moved back to its previous community. This process is applied to each node for its each neighbor in random order and thereby heuristically the quality is tried to be increased.
2) The algorithm constructs a reduced network whose nodes are the communities that are evolved in the first step. Moreover, the weights of the edges in this reduced network are given by the sum of weights of the links between the nodes which reside in the corresponding two communities. Links between nodes of the same community in the old network are presented as self-links for the node that represents that community in the new reduced network. When this reduced network is fully constructed, then algorithm calls itself recursively and first step is applied to this reduced network.

The algorithm keeps recursing until no further improvement in modularity is measured and thereby there are no changes in the community structure. [7]

Louvain algorithm detects community structures whose modularity values are locally optimal with respect to community merging, but not necessarily locally optimal with respect to individual node movements. Since the Louvain algorithm applies local moving heuristic in the beginning of its recursive block and merges communities by reducing network in the end of its recursive block, calling it iteratively ensures that the resulting community structure cannot be improved further either by merging communities or by moving individual nodes from one community to another. Like the iterative variant of these algorithms SLM algorithm constructs community structures that are locally optimal with respect to both individual node movements and community merging. Besides these capabilities, SLM also tries to optimize modularity by splitting up communities and moving sets of nodes between communities. This is done by changing the way that local moving heuristic and network reduction runs.[9]

Louvain algorithm runs local moving heuristic algorithm on the present network as the first step, and then construct the reduced network as the second step. However, the SLM algorithm changes the reduced network construction step by applying following processes:

1) It iterates over all communities that are formed by the first step. It copies each community and constructs a subnetwork that contains only the specific community's nodes.
2) It then runs the local moving heuristic algorithm on each subnetwork after assigning each node in the subnetwork to its own singleton community.
3) After local moving heuristic constructs a community structure for each subnetwork, the SLM algorithm creates the reduced network whose nodes are the communities detected in subnetworks. The SLM algorithm initially defines a community for each subnetwork. Then, it assigns each node to the community that is defined for the node's subnetwork. Thus, there is a community defined for each subnetwork and detected communities in subnetworks are placed under these defined communities as nodes in the reduced network.

This is the way that the SLM algorithm constructs the reduced network. After these processes, the SLM algorithm gives the reduced network to the recursive call as input and all the processes starts again for the reduced network. The recursion continues until a network is constructed that cannot be reduced further. To sum up, the SLM algorithm has more freedom in trying to optimize the modularity by having the ability to move sets of nodes between communities which cannot be done by Louvain algorithm. [9]

B. Dynamic Smart Local Moving Algorithm

SLM algorithm initially assigns each node to a different community, so each node has its own singleton community. In order convert SLM to dynamic form, we replace that operation with a newly defined procedure, called initialize communities which is given in Figure 1. This procedure works as follows:

Procedure: Initialize Communities
Input: Old_Communities, Old_Network, New_Network
Output: New_Communities

1: $j = 0$
2: **for** $i = 0 \rightarrow Old\_Communities.size$ **do**
3: $\quad New\_Communities = ReadCommunitiesFile()$
4: **end for**
5: $Delta\_Network = New\_Network - Old\_Network$
6: **for** $j = i \rightarrow Delta\_Network.size$ **do**
7: $\quad new\_communities[j] = Delta\_Network[j - i]$
8: **end for**

Fig. 1. Initialize Communities Procedure

- Existing communities are read from file as New Communities.
- If exists, the extensions to the network has been determined.
- For each new node, singleton new communities are constructed and added to New Communities.

The effects of other changes in the network, such as adding new edges and deletions of nodes and edges, are handled while executing standard SLM procedure. The new dSLM is available at https://github.com/mertozer/dSLM.

Since after some iterations, the increase of modularity drops to very small values, it might make sense to stop the iterations using either the amount of changes or by setting a target modularity value. We have implemented the second option, which is called dSLMEVS in the experiments. We have made the tests by setting the modularity values as the one obtained for SLM in order to be able to observe the differences in the execution times for exactly the same modularity values.

C. Running Example

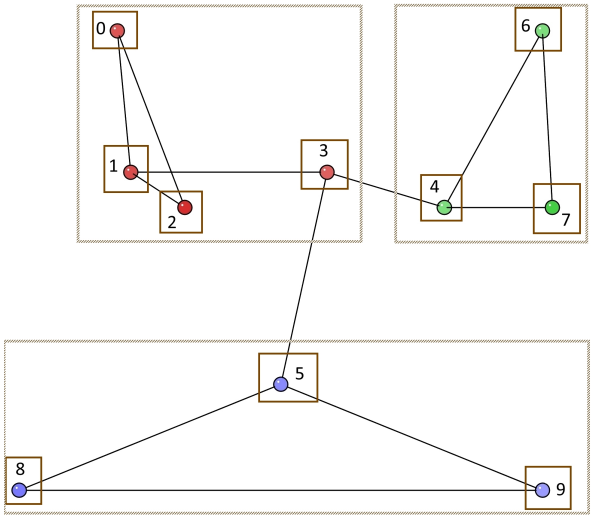

Fig. 2. Network in time t (analyzed by SLM)

Let the sample network depicted in Figure 2 to be a network that changes in time and needs to be analyzed continuously in each time frame. So, the network is analyzed and communities are detected in time t. Figure 2 presents the

beginning and end states of the community structure of the network analyzed by SLM algorithm in time t. Solid rectangles present the initial community structure; whereas the colors of the nodes (and dashed rectangles) present the resulting community structure. From time t to time t+1, a node which is numbered as 10 and an edge between this new node and the node which is numbered as 9 are added to the network. This evolved network in time t+1 can be seen in Figure 3 and Figure 4. Figure 3 presents the community detection process

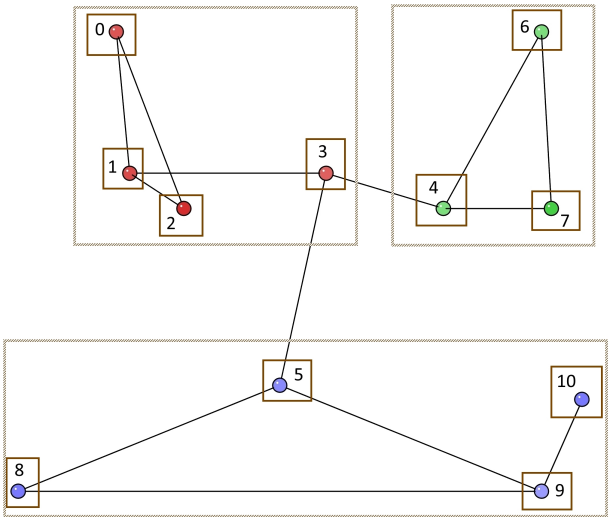

Fig. 3. Network in time t+1 (analyzed by SLM)

of the network in time t+1 performed by SLM algorithm in the same way as Figure 2. As the difference of Figure 2 and Figure 3, a new node and a new edge are only seen in Figure 3. Since they both demonstrate the SLM process, the initial communities are singleton. Figure 4 demonstrates the

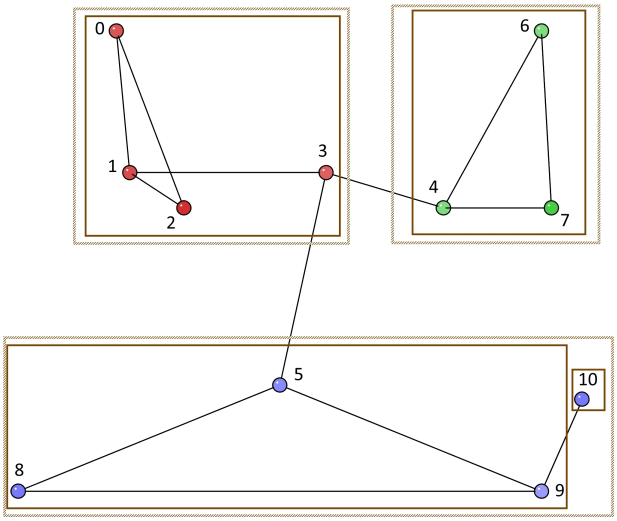

Fig. 4. Network in time t+1 (analyzed by dSLM)

dSLM process of the network in time t+1. In this process, the community structure of the network in time t is used as the initial states of communities which can be seen as rectangles in Figure 4. Both SLM and dSLM algorithms place the newly added node in blue community. SLM constructs the community structure from scratch by trying and finding node movements that maximize the modularity of the network. However, dSLM needs to try only one node movement which is to move newly added node from its singleton community to its only neighbor (blue) community. Since it appears to increase the modularity of the network, dSLM places the new node to blue community and that is it. Because the initial community structure is known to be the one that maximizes the modularity of the network, there is no other node movement trying that can increase modularity. By this way, the dSLM runs faster than SLM . This run time difference between SLM and dSLM gets much greater while the network size increases.

V. Experiments & Results

We evaluate our proposed approach dSLM on five real-world datasets which are the arXiv citation dataset, the GSM calls dataset, Google Plus, Twitter and Youtube user network datasets..

The arXiv[1] citation dataset is published in the KDD Cup 2003. It contains approximately 29,000 papers and their citation graph. In this graph, each vertex represents a paper and each edge represents the citation between its connected vertexes. There are around 350,000 edges which represent citations in this graph.

We have used call detail record (CDR) dataset, obtained from one of the largest GSM operators in Turkey. This produced GSM calls dataset contains 12,521,352 nodes and 44,768,912 edges. These two datasets are used for edge deletion and addition experiments purposes.

The Google Plus and Twitter user network data is collected by Stanford Network Analysis Project[2]. The Google Plus data consists of 107,614 nodes and 13,673,453 edges. The Twitter data consists of 81,306 nodes and 1,768,149 edges. The Youtube user network data is provided by Mislove et al. [32]. It consists of 1,134,890 nodes and 2,987,624 edges. The users of the Youtube are the nodes, and the friendships are represented by edges. We used these 3 datasets for node deletion and addition experiments purposes.

In the first set of experiments we have assumed the dynamic feature is in the form of edge insertions and deletions. Table I and II gives the results for these experiements. In the second set, on the other hand, node insertions and deletions represent the dynamic feature. Table III and IV presents the results for these experiments.

TABLE I. The effect of dSLM for edge insertions

Algorithm	Dataset	Base (# of Edges)	# of Edges Added	Change in Modularity Value	Decrease in Running Time
dSLM	arxiv	300,000	1,000	0.02% increased	26%
dSLM	arxiv	300,000	10,000	0.15% increased	26%
dSLM	GSM	10,000,000	1,000	no change	27%
dSLM	GSM	10,000,000	10,000	no change	29%
dSLM	GSM	10,000,000	100,000	no change	20%
dSLM	GSM	10,000,000	1,000,000	0.02% increased	17%
dSLMEVS	GSM	10,000,000	1,000	no change	91%
dSLMEVS	GSM	10,000,000	10,000	no change	63%
dSLMEVS	GSM	10,000,000	100,000	no change	64%
dSLMEVS	GSM	10,000,000	1,000,000	no change	80%

[1] http://www.cs.cornell.edu/projects/kddcup/datasets.html
[2] http://snap.stanford.edu/data

TABLE II. THE EFFECT OF DSLM FOR EDGE DELETIONS

Algorithm	Dataset	Base (# of Edges)	# of Edges Deleted	Change in Modularity Value	Decrease in Running Time
dSLM	arxiv	300,000	1,000	0.08% increased	32%
dSLM	arxiv	300,000	10,000	0.04% increased	7%
dSLM	GSM	10,000,000	1,000	no change	38%
dSLM	GSM	10,000,000	10,000	no change	27%
dSLM	GSM	10,000,000	100,000	0.01% increased	24%
dSLM	GSM	10,000,000	1,000,000	0.01% increased	16%
dSLMEVS	GSM	10,000,000	1,000	no change	92%
dSLMEVS	GSM	10,000,000	10,000	no change	61%
dSLMEVS	GSM	10,000,000	100,000	no change	91%
dSLMEVS	GSM	10,000,000	1,000,000	no change	80%

TABLE III. THE EFFECT OF DSLM FOR NODE INSERTIONS

Algorithm	Dataset	Base (# of Nodes)	# of Nodes Added	Change in Modularity Value	Decrease in Running Time
dSLM	Twitter	81,296	10	no change	67%
dSLM	Twitter	81,206	100	no change	90%
dSLM	Twitter	80,306	1,000	no change	89%
dSLM	Twitter	71,306	10,000	0.04% increased	70%
dSLM	GPlus	107,604	10	0.02% decreased	72%
dSLM	GPlus	107,514	100	no change	53%
dSLM	GPlus	106,614	1,000	no change	54%
dSLM	GPlus	97,614	10,000	0.97% decreased	11%
dSLM	Youtube	1157728	100	1.77% increased	73%
dSLM	Youtube	1156828	1000	1.36% increased	65%
dSLM	Youtube	1147828	10000	0.03% increased	77%
dSLM	Youtube	1057828	100000	0.12% increased	41%
dSLMEVS	Twitter	81,296	10	no change	82%
dSLMEVS	Twitter	81,206	100	no change	90%
dSLMEVS	Twitter	80,306	1,000	no change	79%
dSLMEVS	Twitter	71,306	10,000	no change	75%
dSLMEVS	GPlus	107,604	10	0.02% decreased	70%
dSLMEVS	GPlus	107,514	100	no change	83%
dSLMEVS	GPlus	106,614	1,000	no change	83%
dSLMEVS	GPlus	97,614	10,000	0.04% increased	30%
dSLMEVS	Youtube	1,157,728	100	0.04% increased	92%
dSLMEVS	Youtube	1,156,828	1000	0.13% increased	99%
dSLMEVS	Youtube	1,147,828	10000	0.08% increased	98%
dSLMEVS	Youtube	1,057,828	100000	0.15% increased	98%

TABLE IV. THE EFFECT OF DSLM FOR NODE DELETIONS

Algorithm	Dataset	Base (# of Nodes)	# of Nodes Added	Change in Modularity Value	Decrease in Running Time
dSLM	Twitter	81,306	10	0.06% increased	85%
dSLM	Twitter	81,306	100	no change	79%
dSLM	Twitter	81,306	1,000	0.02% increased	87%
dSLM	Twitter	81,306	10,000	0.03% decreased	28%
dSLM	GPlus	107,614	10	0.35% decreased	90%
dSLM	GPlus	107,614	100	0.37% decreased	78%
dSLM	GPlus	107,614	1,000	0.35% decreased	75%
dSLM	GPlus	107,614	10,000	0.35% decreased	73%
dSLM	Youtube	1,157,828	100	0.28% decreased	88%
dSLM	Youtube	1,157,828	1000	0.03% decreased	87%
dSLM	Youtube	1,157,828	10000	0.21% decreased	73%
dSLM	Youtube	1,157,828	100000	0.09% decreased	73%
dSLMEVS	Twitter	81,306	10	0.05% increased	92%
dSLMEVS	Twitter	81,306	100	0.02% decreased	81%
dSLMEVS	Twitter	81,306	1,000	0.02% increased	87%
dSLMEVS	Twitter	81,306	10,000	0.05% decreased	28%
dSLMEVS	GPlus	107,614	10	0.36% decreased	90%
dSLMEVS	GPlus	107,614	100	0.35% decreased	78%
dSLMEVS	GPlus	107,614	1,000	0.36% decreased	84%
dSLMEVS	GPlus	107,614	10,000	0.34% decreased	69%
dSLMEVS	Youtube	1,157,828	100	0.29% decreased	99%
dSLMEVS	Youtube	1,157,828	1000	0.01% decreased	99%
dSLMEVS	Youtube	1,157,828	10000	0.19% decreased	99%
dSLMEVS	Youtube	1,157,828	100000	0.07% decreased	98%

In general, dSLM does not decrease the number iterations of convergence of SLM, however, it decreases the number of node movements needed in each iteration of SLM. This indicates that each iteration of dSLM runs faster than each iteration of SLM. Therefore, overall running time of dSLM is less than SLM's overall execution time. The overall results can be seen in Table I, II, III and IV.

In order to be able to decrease the overall running time of dSLM algorithm even more, we added another parameter called expected modularity value that enables the algorithm stop when it is reached. We named this kind of new algorithm as dSLMEVS and made same experiments on it with this new parameter set to the modularity value resulted from SLM algorithm. By this new algorithm and parameter, we aimed to decrease running time as much as possible while keeping the modularity value unchanged or increased. We reached our aim and decreased running time drastically and keep modularity value unchanged or increased as seen in all of the tables.

VI. CONCLUSION

Waltman & Van Eck proposed and implemented the SLM algorithm in order to detect communities in large networks. We extended their implementation to define the community structure in a dynamic rather than static way. We made use of the past calculation results of the SLM algorithm in order to calculate the current networks community structure. This usage is the main extension and contribution to the SLM algorithm. In the basics, it is what extends the SLM to be dSLM.

To sum up, we extended SLM to be incremental and dynamic by using the historical results of community detection algorithms for the initial community assignments of the nodes. Thus, the number of node movement actions tried to maximize the modularity value is decreased. This led to decrease in running time of the algorithms. Moreover, it can lead to decrease in number of iterations to converge. Thus, if the algorithms run with a constant number of iterations parameter, the modularity value may result as increased.

ACKNOWLEDGMENT

This research was supported partially by USAF Grant FA9550-15-1-0004.

REFERENCES

[1] A. Clauset, M. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, p. 066111, 2004. [Online]. Available: http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:cond-mat/0408187

[2] R. Guimera, M. Sales-Pardo, and L. Amaral, "Modularity from fluctuations in random graphs and complex networks," *Physical Review E*, vol. 70, no. 2, p. 025101, 2004.

[3] J. Duch and A. Arenas, "Community detection in complex networks using extremal optimization," *Physical Review E*, vol. 72, p. 027104, 2005. [Online]. Available: http://www.citebase.org/abstract?id=oai:arXiv.org:cond-mat/0501368

[4] M. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical Review E*, vol. 74, no. 3, p. 36104, 2006.

[5] S. Lehmann and L. K. Hansen, "Deterministic modularity optimization," *The European Physical Journal B*, vol. 60, no. 1, pp. 83–88, 2007. [Online]. Available: http://dx.doi.org/10.1140/epjb/e2007-00313-2

[6] J. Lee, S. P. Gross, and J. Lee, "Mod-csa: Modularity optimization by conformational space annealing," *CoRR*, vol. abs/1202.5398, 2012.

[7] V. Blondel, J. Guillaume, R. Lambiotte, and E. Mech, "Fast unfolding of communities in large networks," *J. Stat. Mech*, p. P10008, 2008.

[8] R. Rotta and A. Noack, "Multilevel local search algorithms for modularity clustering." *ACM Journal of Experimental Algorithmics*, vol. 16, 2011.

[9] L. Waltman and N. J. van Eck, "A smart local moving algorithm for large-scale modularity-based community detection." *CoRR*, vol. abs/1308.6604, 2013.

[10] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner, "On modularity clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 2, pp. 172–188, 2008.

[11] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, L. Liberti, and S. Perron, "Column generation algorithms for exact modularity maximization in networks," *Physical Review E*, vol. 82, no. 4, article, pp. –, jan 2010.

[12] G. Xu, S. Tsoka, and L. G. Papageorgiou, "Finding community structures in complex networks using mixed integer optimisation," *The European Physical Journal B*, vol. 60, no. 2, pp. 231–239, 2007. [Online]. Available: http://dx.doi.org/10.1140/epjb/e2007-00331-0

[13] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, p. 066111, Dec 2004. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.70.066111

[14] P. Holme and J. Saramäki, "Temporal networks," *Physics Reports*, vol. 519, no. 3, pp. 97–125, 2012.

[15] T. Yang, Y. Chi, S. Zhu, Y. Gong, and R. Jin, "Detecting communities and their evolutions in dynamic social networks - a bayesian approach." *Machine Learning*, vol. 82, no. 2, pp. 157–189, 2011.

[16] X. Tang and C. C. Yang, "Dynamic community detection with temporal dirichlet process." in *SocialCom/PASSAT*. IEEE, 2011, pp. 603–608.

[17] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '06. New York, NY, USA: ACM, 2006, pp. 554–560. [Online]. Available: http://doi.acm.org/10.1145/1150402.1150467

[18] M.-S. Kim and J. Han, "A particle-and-density based evolutionary clustering method for dynamic networks." *PVLDB*, vol. 2, no. 1, pp. 622–633, 2009.

[19] D. Greene, D. Doyle, and P. Cunningham, "Tracking the evolution of communities in dynamic social networks." in *ASONAM*, N. Memon and R. Alhajj, Eds. IEEE Computer Society, 2010, pp. 176–183.

[20] P. Brodka, S. Saganowski, and P. Kazienko, "Group evolution discovery in social networks," in *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*, 2011, pp. 247–253.

[21] T. Dinh, Y. Xuan, and M. Thai, "Towards social-aware routing in dynamic communication networks," in *Performance Computing and Communications Conference (IPCCC), 2009 IEEE 28th International*, Dec 2009, pp. 161–168.

[22] T. Aynaud and J.-L. Guillaume, "Static community detection algorithms for evolving networks," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt), 2010 Proceedings of the 8th International Symposium on*, May 2010, pp. 513–519.

[23] M. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[24] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, pp. 75–174, 2010.

[25] M. E. J. Newman, "Analysis of weighted networks," *Phys. Rev. E*, vol. 70, no. 5, p. 056131, Nov. 2004. [Online]. Available: http://pre.aps.org/abstract/PRE/v70/i5/e056131

[26] A. Arenas, J. Duch, A. Fernandez, and S. Gmez, "Size reduction of complex networks preserving modularity," *CoRR*, vol. abs/physics/0702015, 2007.

[27] E. A. Leicht and M. E. J. Newman, "Community structure in directed networks," *Phys. Rev. Lett.*, vol. 100, no. 11, p. 118703, Mar. 2008. [Online]. Available: http://prl.aps.org/abstract/PRL/v100/i11/e118703

[28] S. Fortunato and M. Barthlemy, "Resolution limit in community detection," *Proceedings of the National Acadamy of Sciences of the United States of America (PNAS)*, vol. 104, no. 1, pp. 36–41, 2007.

[29] J. Reichardt and S. Bornholdt, "Statistical mechanics of community detection," *Arxiv preprint cond-mat/0603718*, 2006.

[30] V. A. Traag, P. Van Dooren, and Y. Nesterov, "Narrow scope for resolution-limit-free community detection," *Physical Review E*, vol. 84, no. 1, p. 016114, 2011.

[31] L. Waltman, N. J. van Eck, and E. C. Noyons, "A unified approach to mapping and clustering of bibliometric networks," *Journal of Informetrics*, vol. 4, no. 4, pp. 629–635, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/B83WV-50RFN28-1/2/809f89176e8076cac3862b6589bc6fd5

[32] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and Analysis of Online Social Networks," in *Proceedings of the 5th ACM/Usenix Internet Measurement Conference (IMC'07)*, San Diego, CA, October 2007.