Graph Partitioning in Parallelization of Large Scale Networks

Sima Das*, Jennifer Leopold*, Susmita Ghosh†, Sajal K. Das*
*Department of Computer Science
Missouri University of Science and Technology
Rolla, Missouri, USA 65409.
sdp4b; leopoldj; sdas@mst.edu

†Department of Computer Science and Engineering
Jadavpur University
Jadavpur, Kolkata, INDIA
sghosh@cse.jdvu.ac.in; susmitaghoshju@gmail.com

***Abstract*—Real world large scale networks exhibit intrinsic community structure, with dense intra-community connectivity and sparse inter-community connectivity. Leveraging their community structure for parallelization of computational tasks and applications, is a significant step towards computational efficiency and application effectiveness. We propose a *weighted depth-first-search* graph partitioning algorithm for community formation that preserves the needed community dependency without any cycles. To comply with heterogeneity in community structure and size of the real world networks, we use a flexible limiting value for them. Further, our algorithm is a diversion from the existing modularity based algorithms. We evaluate our algorithm as the quality of the generated partitions, measured in terms of number of graph cuts.**

I. Introduction

Large scale networks are an integral part of our daily life, be it the Internet, biological network, social network and gene regulatory network, to mention a few. Irrespective of their widely varying application domains, they resemble similar network structural properties, for example: scale-free, small-world property, small diameter, neighborhoods of surprisingly dense structure irrespective of relatively sparse over all network, large connected component, nearly power-law degree distribution etc. [6], [13]. These structural properties are leveraged in interesting ways for computational efficiency and effectiveness of range of applications and computational tasks [4], [3], [11].

One of the significant structural property, exhibited by these wide range of diverse large scale networks is that, they exhibit community structure, i.e. a set of nodes having more and/or better connections between its members than with the remainder of the network. We propose to consider these communities/partitions as the basic elements for our parallel execution. The partitioning schemes, partition the graph into predefined number of parts of equal size. The move-based algorithms for graph partitioning problem iteratively try to improve the partition, by vertex moves or swaps between the partitions created during iterations, such as Kernighan-Lin algorithm (KL) [8]. The algorithm converges to a local optimum by selecting moves that minimize the cost of graph cut. n the multilevel algorithms, the input graph is coarsened iteratively by merging vertices according to a matching, until a small graph with similar structure is generated. This graph can then be partitioned with spectral method, or greedy graph growing algorithm [14], [7]. In the subsequent phase, the graph is iteratively un-coarsened and the KL algorithm is used in each iteration. In fact, the multilevel scheme is behind the state-of-the-art graph partitioning libraries, such as METIS [7]. But in real world networks, say, in social/biological networks, neither the size, nor the number of clusters can be fixed.

Thus, determining inherent communities (clusters or modules) in large scale networks, is one of the fundamental network analysis problems and is widely studied [12], [9], [6], reflecting dense intra-cluster (connections within a cluster) and sparse-inter cluster (connections across cluster) connections. From here onwards, to refer to a densely connected group of nodes, we will use cluster or community or module interchangeably. Since, finding optimal community structure is known to be NP-hard [5], heuristics and approximation solutions have been proposed based on agglomerative hierarchical clustering [1], [2], divisive clustering based on betweenness centrality [10], or spectral partitioning [14]; though all of them validate their approach by optimizing (maximizing) the modularity index.

Besides, the application tasks like information flow, routing etc., there are computational tasks that can be greatly benefitted from the intrinsic community structure. The work in Das et al., [2], leverages the underlying community structure for efficient computation of betweenness centrality indices. Here, the community detection (formation) acts as a preprocessing stage to the subsequent influential node evaluation and help exploit the divide-conquer algorithmic technique. More so, evaluation of stress centrality, closeness centrality and other similar distance based centrality metrics can be computed in a similar manner.

Moreover, the large scale network graph also justifies the need for parallel computation, besides the distributed approach in [2]. In fact, computation of influential nodes (centrality metric like betweenness centrality) in a network, evolution of network mechanisms say, interactions in gene regulatory networks can be efficiently analyzed through parallel computation. Though, parallel computation also resembles distributed computation in minimizing inter-cluster edge weight (or interprocessor communication), there is also a class of parallel computation framework called multiple instruction multiple

DOI 10.1109/LCN.2016.36

data architecture (MIMD) with shared memory and MIMD architecture without shared memory having number of interprocessor connections larger than the number of processors, both of which does not rely on minimizing interprocessor communication. Thus, it is interesting to investigate creation of these special class of partitions that preserve their desired interdependency yet are cyclically connected, as basic computation elements to the above class of parallel framework. It is also interesting to investigate to what degree they reflect the underlying community structure.

We make the following contribution to address the above issues.

II. Summary of Contributions

Here, we propose a weighted depth-first-search-based (WDFS-based) graph partitioning algorithm for community formation/detection, that preserves the required dependency among communities, yet the connectivity graph over these communities is of acyclic nature. The graph partitioning combinatorial optimization problem is known to be NP-hard [5], thus our proposed heuristic for graph partitioning with flexible bounds on size and number of partitions is based on divide and conquer algorithmic technique.

Our proposed heuristic acts as a preprocessing stage to generate communities (partitions) to be used over computational environments for efficient computation of computational tasks. The generated partitions can be used in a parallel environment that is not critical to interprocessor communication cost for computational tasks, say analyzing temporal evolution, evaluating influential nodes etc. As communication cost is of lesser significance, modularity maximization is not our objective, rather we see how well these dependency preserving, acyclically connected partitions reflect the under lying real world community structure. As modularity index above 0.3 is considered to be good, any generated partitioned structure with modularity higher than this, reflects good resemblance with underlying intrinsic community structure of the networks. In this aspect we examine how the proposed approach varies in modularity index relative to the work in [2], which optimizes modularity index.

To leverage the flexibility in underlying application network structure, as well as allow for the inhomogeneity (heterogeneity) in computational environments, instead of a fixed limit on number and size of communities (partitions), we propose an upper bound on them ($O(\sqrt{|V|})$), where $|V|$ is the number of nodes in the network. This helps in forming communities of different sizes (heterogeneous) and numbers. Further, the intuition of $O(\sqrt{|V|})$, resembles the fact that efficient parallel execution of tasks needs to have similar amount of task distribution, where the tasks are of almost same complexity and the computing elements are equipped with similar computing resources. Further, it also helps achieve good modularity index [2].

Our solution to the partition construction caters to the class of application problems or computational environments, where interprocessor communication cost is not critical. We compare the quality of our resulting partitions relative to the partitions generated using METIS 4.0, in terms of the graph cut size. We also examine the community structure (modularity index) w.r.t. the a specific graph cut size for these approaches.

The rest of the paper is organized as follows. In Section III we present preliminary concepts and definitions used in our work and formally define our problem. Section IV describes our *divide and conquer*-based, weighted depth-first-search graph partitioning algorithm. Section V reports experimental results, and finally conclusions and future works are offered in Section VI.

To the best of our knowledge ours is the first work to consider graph partitioning in parallelization of computational tasks over large scale networks. We propose a divide and conquer-based heuristic that uses weighted depth-first-search tree and post order traversal on the subsequent spanning tree in generating partitions (communities).

III. Preliminary Definitions and Problem Formulation

Let us consider the underlying directed, weighted network graph of any application network, represented as $G(V, E, W)$, where V, E and W are the set of nodes, edges and edge weights respectively. For any pair of nodes $u, v \in V$, directed edge from u *to* v is represented as e_{uv} and its weight is denoted as $w(e_{uv})$. Any undirected graph can be mapped to G with each undirected edge as bidirectional edges.

Let the resulting partitioned graph be represented as $G'(V', E', W')$. The graph G' constitutes the partitions of graph G, with V' as the set of coagulated nodes from V, we call them as *communities or clusters*. Further, for any pair of nodes $u', v' \in V'$, $e'_{u',v'} \in E'$ iff $\exists u \in u', v \in v'$, s.t. $e_{uv} \in E$, thus E' is the set of coagulated edges in G'. We virtually nullify the existence of edges within any super node (i.e. intra-cluster edge), though they exist physically and remain unaffected. Further, $w'(e_{u'v'}) = \sum_{\forall u \in u', v \in v'} w(e_{uv})$, where *super weight* $w' \in W'$.

Let the given input graph be the example, in 1st graph (graph with green dots and edges) of Figure 1. For simplicity, we consider undirected graph. The corresponding coagulated nodes (cluster or communities), resulted from the graph partitioning algorithm, are shown in the 2nd graph of same figure (blue dots). The clusters are 12, 37, 56 and 48. The partitioning algorithm acts as a mapping from the input graph in left to the partitioned graph in right.

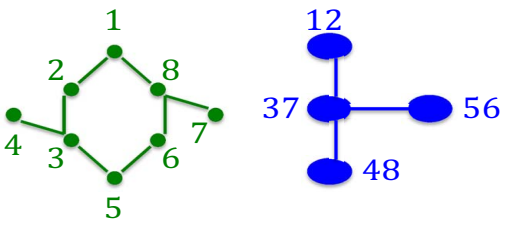

Fig. 1: Partitioned Graph: Example

Here, we introduce the following two constraints as the basis of our problem formulation.

- The number of clusters (communities) in G' is bounded, i.e. $|V'| \leq k$, where k is a positive integer.
- The number of nodes within cluster node v' is also bounded, i.e. $\forall v' \in V'$, $|v'| \leq q$, where q is a positive integer.

We formally define our problem as follows:
Given a network graph G and the positive integers k *and* q, we propose a divide and conquer-based graph partitioning algorithm that results in the partitioned graph G', such that G' satisfies the above two constraints and with modularity index of G' (without constraint on inter-cluster edge weight). Modularity index for partitioned graph is given by $Q_{G'} = \sum_{\forall v' \in V'} Q_{v'}$. Further, there is no other partitioned graph G'', such that $Q_{G''} > Q_{G'}$ and G'' satisfies the above constraints.

We define modularity index for any cluster node $v' \in G'$ similar to [10], $Q_{v'} = \frac{\sum_{\forall e_{uv} \in v'} w'(e_{uv})}{w'(E')} - \frac{\sum_{\forall u' \in V' \setminus v'} w'(e_{v'u'})}{w'(E')}$. This facilitates the resulting community structure of the partitioned graph, thus resembling the intrinsic clustering exhibited by real-world networks, though our partition procedure does not employ inter-cluster edge weight minimization. The numerator of the 1st fraction is the total edge weight within a cluster node v', whereas the numerator of the second fraction is the total edge weight of the cluster node (v') with all other cluster nodes u' (i.e $\forall u' \in V' \setminus v'$). Further, $w'(E')$ is the total edge weight in G'. Thus, each fraction computes the weighted measure.

IV. Graph Partitioning Heuristic

Our proposed graph partitioning algorithm works in two phases. In the 1st phase it creates a *spanning tree* over the input graph using *weighted depth-first-search* (WDFS) traversal. In the second phase, a *post order traversal* is performed over the spanning tree from the 1st phase. This phase *accumulates* nodes as per some rules, in generating the final partitioned graph with coagulated (cluster) nodes and coagulated edges. The partitioned graph resulting after these two phases has coagulated nodes or clusters with the required dependency among cluster connectivity preserved, yet is acyclic. Further, our flexibility of our proposed graph partitioning algorithm results from the $O(\sqrt{|V|})$ constraint on the size and number of communities, instead of a strict or fixed limit.

The WDFS traversal, is a diversion from usual graph theoretic depth-first-search traversal in selecting nodes, though follows the same traversal principle. Any node with non empty children and with more than one of these children being non leaf nodes, it selects the child with *least connecting edge weight* with it. Since, during this traversal procedure, we are intrinsically creating a path between any pair of nodes of the graph, selecting the edges in the described manner helps reduce the inter-cluster edge weight wherever possible. Further, child nodes that are leaf nodes always have the priority over non-leaf child nodes.

The post order traversal is in accordance with the usual graph traversal algorithm. In this phase, any single child of a parent that is also a leaf node, is accumulated with its parent, in creating a super node. Moreover, for a pair of leaf nodes in the post order traversal of the DFS tree, if they share common parent, then these two leaf nodes are accumulated to form a cluster node.

In the spanning tree from 1st phase, the nodes are accumulated, without compromising the two constraints given in the problem formulation, i.e the constraints on size and number of cluster nodes in the resulting partitioned graph.

In the following Subsection we formally describe our algorithm.

A. Algorithm: Graph Partition

We store the input graph in an adjacency list. We use stack as our basic data structure, denoted here as STCK. We also use an auxiliary array, AR. The stack is used to store the nodes added to the spanning tree but not yet coagulated. Nodes of the input graph that are in the spanning tree are crossed out, where as nodes that do not yet belong to the spanning tree are termed New.The top of stack pointer, denoted as TOS. We have another array A used during WDFS traversal, the spanning tree in A is passed as input to $ClusterFormation()$:, accumulation in storing accumulated/coagulated nodes. Array A store for each node id v another node id $A(v)$. Intuitively, it is desirable to join node v with node $A(v)$ if $A(v)$ exists.

We give the steps of our proposed algorithm below.

Creation of Spanning Tree():

1) Input: Network graph G.
2) Initialize data structures. Set node v to be any input node id and put node v on top of STCK.
3) Find a New node u, adjacent to v, giving priority to u, as per the rules of weighted DFS. Put u on top of STCK. While u does not exist, go to Cluster Formation().
4) Find New node m such that (u, m) and (v, m) are edges in the input graph G. Set $A(v) = u, A(u) = m$. Put m on top of STCK. Set $v = u$ and $u = m$.
5) If such m does not exist, then find a pair of New nodes m *and* n, such that $(u, m), (m, n), (n, v)$ are edges in G. Set $A(v) = u, A(u) = m$ *and* $A(m) = n$. Put m *and* n on STCK. Set $v = m, u = n$.
6) If the search for nodes in (3) and (4) is unsuccessful then set $v = u$ and repeat from step (3).
7) Output: The spanning Tree of the nodes.

Cluster Formation():

1) Input: Spanning Tree from Creation of Spanning Tree().
2) Initialize auxiliary array AR to A.
3) While AR(TOS) is not empty, then join nodes i and j, i =A(TOS), and j =AR(TOS) into a pair and call REPEAT(i, j).
4) While AR(TOS) is empty, but AR(TOS-1) is not empty, then join nodes i and j, i =A(TOS), and j =AR(TOS-1) into a pair and call REPEAT(i, j).
5) If both AR(TOS) and AR(TOS-1) are empty, set AR(TOS-1)=A(TOS).
6) Decrement TOS, set v =A(TOS), repeat from step (3).

7) Output: Partitioned graph with Clusters

In procedure REPEAT(i, j), if A$(i) \neq j$ and A(i) is joined into a pair with node u, such that A(A$(i)) \neq u$, then change the partition, so that i is joined with A(i), j is joined with u and the rest of the partition remains unchanged.

The computation cost of the proposed graph compaction algorithm for k compaction is $O(nd \log n)$, where d is maximum node degree.

V. Experimental Results

To evaluate the effectiveness of our proposed WDFS for flexible graph partitioning problem, we examine the resulting partitions or cluster nodes. We consider an upper bound of $O(\sqrt{|V|})$ on number and size of cluster nodes.

We consider four distinct network data sets, NetScience, twitter mention, power grid, and Internet [15]. We give the network statistics of the four networks in table I. In Table II, we show the modularity index with $O(\sqrt{|V|})$ constraint on cluster nodes for both WDFS and Das et al. [2] (Modularity). Modularity index above 0.3 is considered good, thus as it is evident from the tabular values, WDFS-based flexible graph partition reasonably reflects underlying community structure.

TABLE I: Network Statistics: Example Network Data Sets

Network measures over Four Distinct Data Sets				
Network Measure	Internet	Power Grid	Net Science	Twitter Mention
Number of Nodes	22963	4941	1859	3656
Number of edges	48436	6594	2742	157727
Network Density	2.1	1.334	1.329	43.14
Average Degree	4	2	3	86

TABLE II: Modularity Index with Constraint on ClusterNode

Modularity Index over Four Distinct Data Sets				
ModularityIndex: ClusterNode$O(\sqrt{\|V\|})$ Selected Nodes	Internet (210)	Power Grid (80)	Net Science (180)	Twitter Mention (61)
WDFS	0.45	0.53	0.5	0.39
Modularity	0.69	0.81	0.65	0.58

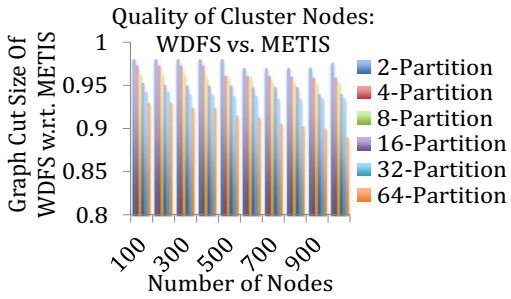

Fig. 2: Partition Quality Evaluation

TABLE III: Modularity Index WDFS vs. METIS

Modularity Index for Twitter over distinct graph cuts				
ModularityIndex: ClusterNode	2-partition	8-partition	32-partition	64-partition
WDFS	0.18	0.23	0.31	0.43
METIS	0.13	0.2	0.25	0.37

We evaluate the partition quality relative to the partitions formed using METIS 4.0. We consider 2 to 64 compaction over Twitter network. We see that in each case our proposed algorithm has 2% to 10% lower graph cuts w.r.t. METIS. This is shown in the Figure 2. For brevity we omit the plot of cut size comparison with Das et al. [2] (Modularity). We also show the modularity index corresponding to the cut sizes for the two approaches in Table III. The cluster nodes are input to the parallel execution stage for subsequent efficient execution of computational tasks.

VI. Conclusion

It is interesting to observe that the resulting partitioned graph with communities, can be visualized as the compact graph of the original graph. It has communities as its nodes, which will be considered as basic computing elements for efficient parallel execution of computational tasks. We propose to prove the asymptotic optimality of our resulting partitioned graph. We would also like to give extensive investigation results reflecting the purpose of constraints in the proposed approach, and explore for graph cuts and community formation relative to METIS and Modularity. We would also like to investigate the computational cost reduction in parallel execution of task over these cluster units and finally aggregating the results wherever possible, which exploits the divide and conquer algorithmic technique.

References

[1] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, pp. 1– 6, 2004.

[2] S. Das and S. K. Das, "Leveraging network structure in centrality evaluation of large scale networks," in *40th IEEE Conference on Local Computer Networks (LCN)*, 2015, pp. 579–586.

[3] Y. Dong, J. Zhang, J. Tang, N. V. Chawla, and B. Wang, "Coupledlp: Link prediction in coupled networks," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 199–208.

[4] W. Gao, Q. Li, and G. Cao, "Forwarding redundancy in opportunistic mobile networks: Investigation and elimination," in *Proceedings of The IEEE INFOCOM*, 2014, pp. 2301–2309.

[5] M. R. Garey and D. S. Johnson, *Computers and intractability*. wh freeman New York, 2002, vol. 29.

[6] 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.

[7] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[8] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[9] M. E. J. Newman, "The structure and function of complex networks," *SIAM REVIEW*, vol. 45, pp. 167–256, 2003.

[10] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, Feb. 2004.

[11] E. Sherkat, M. Rahgozar, and M. Asadpour, "Structural link prediction based on ant colony approach in social networks," *Physica A: Statistical Mechanics and its Applications*, vol. 419, pp. 80–94, 2015.

[12] S. H. Strogatz, "Exploring complex networks," *Nature*, vol. 410, no. 6825, pp. 268–276, 2001.

[13] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow, "The anatomy of the facebook social graph," *arXiv preprint arXiv:1111.4503*, 2011.

[14] S. White and P. Smyth, "A spectral clustering approach to finding communities in graphs," in *In SIAM International Conference on Data Mining*, 2005.

[15] G. Wiki. (2014) Dataset for our experiment. [Online]. Available: https://wiki.gephi.org/index.php/Datasets