FlowPro: A Flow Propagation Method for Single Community Detection

Costas Panagiotakis*, Harris Papadakis†, and Paraskevi Fragopoulou†
*Department of Business Administration,
Technological Educational Institute of Crete, Agios Nikolaos, Greece
Email: cpanag@staff.teicrete.gr
†Department of Informatics Engineering,
Technological Educational Institute of Crete, Heraklion, Greece
Email:adanar@ie.teicrete.gr, fragopou@ics.forth.gr

***Abstract*—In this paper, we propose a flow propagation algorithm (FlowPro) that finds the community of a node in a complex network. The novelty of the proposed approach is the fact that FlowPro is local and it does not require the knowledge of the entire graph as most of the existing methods from literature. This makes possible the application of FlowPro in extremely large graphs or in cases where the entire graph is unknown like in most social networks. In addition, it simply compute the community of exactly one node, that is the major issue in most social network based applications. Experimental results and comparisons with other methods from the literature are presented for a variety of benchmark graphs with known community structure, derived by varying a number of graph parameters and real dataset graphs.**

I. Introduction

Various applications like finding web communities, detecting the structure of social networks, or even analyzing a graph's structure to uncover Internet attacks are just some of the applications for which community detection is important. Several attempts have been made to provide a formal definition to the generally described "community finding" concept, providing different approaches. Some of them aim at detecting the so-called, *strong communities*, groups of nodes for which each node has more edges to nodes of the same community than to nodes outside the community [1]. Others aim at detecting *weak communities*, which is defined as a subgraph in which the sum of all node degrees within the community is larger than the sum of all node degrees towards the rest of the graph [2]. Variations also appear in the method used to identify communities: Some algorithms follow an iterative approach starting by characterizing either the entire network, or each individual node as community, and splitting [3] or merging [2] communities respectively, producing a hierarchical tree of nested communities, called *dendrogram*. Several researchers aim to find the entire hierarchical community dendrogram [3] while others wish to identify only the optimal community partition [1]. More recently used approaches aim to identify the community surrounding one or more seed nodes [4]. Some researchers aim at discovering distinct (non-overlapping) communities, while others allow for overlaps between communities [5].

Paraskevi Fragopoulou is also with the Foundation for Research and Technology-Hellas, Institute of Computer Science, 70013 Heraklion, Crete, Greece.

Two comprehensive and relatively recent surveys covering the latest developments in the field can be found in [6], [7]. While the first algorithms for the problem used the agglomerative approach trying to derive an optimal community partition by merging or splitting other communities, recent efforts concentrate on the derivation of algorithms based exclusively on local interaction between nodes. A community surrounding a seed node is identified by progressively adding nodes and expanding a small community.

One of the most known community finding algorithms was developed by Girvan an Newman [3]. This algorithm iteratively removes edges participating in many shortest paths between nodes (indicating bridges), connecting nodes in different communities. By gradually removing edges, the graph is split and its hierarchical community structure is revealed. The algorithm is computationally intensive because following the removal of an edge, the shortest paths between all pairs of nodes have to be recalculated. However, it reveals not only individual communities, but the entire hierarchical community dendrogram of the graph.

The authors of [4] introduce a local methodology for community detection, named *Bridge Bounding*. The algorithm can identify individual communities starting at seed nodes. It initiates community detection from a seed node and progressively expands a community trying to identify *bridges*. An edge is characterized as a bridge by computing a function related to the *edge clustering coefficient*. The edge clustering coefficient is calculated for each edge, looking at the edge's neighborhood, and edges are characterized as bridges depending on wether their clustering coefficient exceeds a threshold. The method is local, has low complexity and allows the flexibility to detect individual communities, albeit less accurately. Additionally, the entire community structure of a network can be uncovered starting the algorithms at various unassigned seed nodes, till all nodes have been assigned to a community.

Another efficient algorithm is the one described by Chen et al. in [8]. The algorithm follows a top down approach where the process starts with the entire graph and sequentially removes inter-community links (bridges) until either the graph is partitioned or its density exceeds a certain desired threshold. If a graph is partitioned, the process is continued recursively on its two parts. In each step, the algorithm removes the link between two nodes with the smallest number of common neighbors. The density of a graph is defined as the number of edges in the graph divided by the number of edges of a complete graph with the same number of nodes.

The algorithm described by Blondel et al. in [9] follows a bottom-up approach. Each node in the graph comprises a singleton community. Two communities are merged into one if the resulting community has larger modularity value [10] than both the initial ones. This is a rapid and accurate algorithm which detects all communities in the graph. In suffers however, in the sense, from the fact that during its execution, it constantly requires the knowledge of some global information of the graph, namely the number of its edges (which changes during the execution since the algorithm modifies the graph), limiting, to a certain extend, its distributed nature.

In [11]–[14] a distributed community detection algorithm that identifies the entire community structure of a network based on interactions between neighboring nodes. In the core of our proposal lies the spring metaphor which inspired the Vivaldi synthetic network coordinate algorithm [15]. Extensive experiments on several benchmark graphs with known community structure indicate that our algorithm is highly accurate in graph partitioning into non-overlapping communities, when the entire graph structure is given. In addition, this method has been successfully applied on interactive image segmentation problem [16] resulting high performance results and outperforming other methods from literature.

We will now describe a state-of-the-art algorithm that we compare our approach with, in the experimental evaluation section. Lancichinetti et al. algorithm [5] is able to detect a community starting on a one given node without the knowledge of the entire graph similarly to our proposed Flow Propagation method (FlowPro). In addition, this algorithm is developed based on the observation that network communities may have overlaps, and thus, algorithms should allow for the identification of overlapping communities. Based on this principle, a local algorithm is devised developing a community from a starting node and expanding around it based on a *fitness* measure. This fitness function depends on the number of inter- and intra-community edges and a tunable parameter α. Starting at a node, at each iteration, the community is either expanded by a neighboring node that increases the community fitness, or shrinks by omitting a previously included node, if this action results in higher fitness for the resulting community. The algorithm stops when the insertion of any neighboring node would lower the fitness of the community. This algorithm is local, and able to identify individual communities. The entire overlapping and hierarchical structure of complex networks can be found by initiating the algorithm at various unassigned nodes.

Most of the approaches found it the literature are centralized, heuristic without a global optimality criterion. At the same time they require the entire graph structure partitioning the entire given graph into overlapping or non-overlapping communities. On the contrary, in this paper, we have proposed a local algorithm that can be implemented as a fully distributed method that solves the single community detection problem without the knowledge of the entire graph structure. This makes possible the application of FlowPro in extremely large graphs or in cases where the entire graph is unknown like in most social networks. According to the problem formulation of FlowPro, the probability of a node to belong on the requested community is analogous on its stored flow. So, the stored flow can be used as a belief (rating) of a node to belong on the community and to answer on several questions like find the k-nearest neighbors of the community of a node that are important on several applications of social networks. Lancichinetti et al. algorithm also is able to solve the single community detection problem, but there is no any requirement that the initial given node will also belongs on the detected community. In addition, another strong point of the proposed method is that according to the experimental results and comparisons on real and synthetic data sets, the proposed method clearly outperform the to Lancichinetti et al. algorithm.

In each iteration of the main process of FlowPro, the initial node propagates a flow that is shared to its neighbors. Each node is able to store, propagate to its neighbors and return a part of flow to the initial node. When the algorithm converges, the stored flow of the nodes that belong on the community of initial node is generally higher than the stored flow of the rest graph nodes resulting the requested community. A flow propagation algorithm that has been successfully used on point clustering problem [17] is the affinity propagation (AP) method [18]. AP takes as input measures of similarity between pairs of data points using negative squared error solving the entire community detection problem. Real-valued messages are exchanged between data points until a high-quality set of exemplars and corresponding clusters gradually emerges. The number of clusters is automatically estimated by the AP, influenced by the values of the input preferences, but also emerges from the message-passing procedure.

The remaining of the paper is organized as follows: Section II presents the problem formulation. Our community detection algorithm is presented and analyzed in Section III. Section IV describes the experimental framework and comparison results with other known algorithms on a number of benchmark graphs. Finally, we conclude in Section V with some directions for future research.

II. Problem Formulation

This section, presents the local community detection problem and studies some issues that have been taken into account in the proposed algorithm. Let $G = (V, W)$ denote the given graph comprising a set V of nodes together with a set W of edges weights. In order to simplify the problem definition we suppose that the graph is undirected and the edges' weights are equal to one if the edge exists. So, if there exist an edge

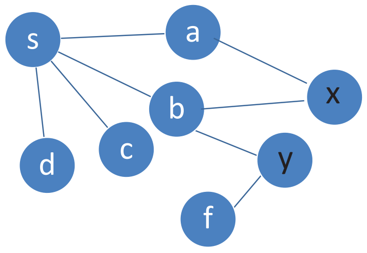


Fig. 1: A simple example of graph.

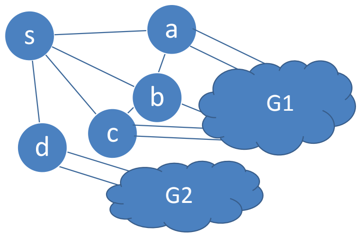


Fig. 2: An example of graph with two almost full connected subgraphs G1 and G2.

from node $i \in V$ to node $j \in V$, then the edge weight is given by $W(i,j) = 1$, otherwise $W(i,j) = 0$. The proposed problem formulation as well as the proposed method can be extended to undirected weighting graphs.

According to the problem definition of local community detection, the initial node ($s \in V$) is given and the goal is to find the set of nodes $C(s)$ that belong to the community of s, with $C(s) \supseteq \{s\}$. This means that there exists high number of edges between the nodes of $C(s)$ comparing with the number of edges that connects the nodes of $C(s)$ and the rest graph.

Let $p(x)$, $x \in V$ denotes the probability that node x belongs on $C(s)$. Let $d(x)$, $x \in V$ denotes the shortest path distance between the nodes x and s. Let $\{e_1, e_2, \cdots, e_{d(x)}\}$ be the set of edges of shortest path between the nodes s and x. Then, $p(x)$ can be estimated by the probability that the $e_1 \in C(s) \wedge e_2 \in C(s) \cdots, e_{d(x)} \in C(s)$. Let ρ be the average ratio of of local links to node degree value, which dictates how strong the clear the communities in the graph are. So, if we ignore the graph structure then the probability of an edge e_1 to belong on $C(s)$ is given by ρ. Therefore, a simple estimation of $p(x)$ can be given by Equation 1 under the assumption that the possibilities $e_i \in C(s)$ and $e_{i+1} \in C(s)$ are independent.

$$p(x) = \rho^{-d(x)} \tag{1}$$

This estimation ignores the graph structure and it takes into account only the $d(x)$.

In addition, we have assumed that the Equation 2 is true for $p(x)$.

$$p(x) \leq \frac{\sum_{y \in n(x)} p(y)}{|n(x)|} \tag{2}$$

where $n(x)$ denote the set of neighbors of node x and $|n(x)|$ denote the number of neighbors of node x. We have used inequality instead of equality in order to take into account the case of bridge, meaning that it is possible that x does not belong on $C(s)$ even if x is connected with s. The proposed algorithm is based on the Equations 1 and 2 in order to estimate a quantity $S(x)$ that is analogous to $p(x)$. So, according to the problem formulation of FlowPro, the probability of the node x to belong on the community of node s is analogous on its stored flow $S(x)$.

III. Flow Propagation Algorithm

In this section, the proposed community finding algorithm (FlowPro) is presented. FlowPro requires as input the initial node ($s \in V$) that search for its community that has an initial flow for propagation $T(s) = |n(s)|$. In each iteration of the main process of FlowPro, the initial node propagates a flow that is shared to its neighbors. Each node is able to store, propagate to its neighbors and return a part of flow to the initial node. Hereafter, we give a detailed description of FlowPro algorithm that it comprises the following steps:

- In each iteration of the main process, the initial node s propagates a flow that is shared to its neighbors according to the edge weights. Each node that receives a flow stores the half, and then send the half of the flow to its neighbors only if the flow is greater than a threshold in order to be able to terminate the process. So, when the graph is undirected without edge weights, the flow is equally distributed to the neighbors, since the edges are equivalent and the node x supposes that the probability of each of the neighbors to belong on $C(s)$ is equal. Since $s \in C(s)$, the flow propagation is executed under the assumption that the node s does not receive/store flow meaning that we set $W(x,s) = 0$, $\forall x \in V$ (see line 5 of Algorithm 1).
  Let $T(x)$ be the quantity of flow that the node $x \in V$ is going to transmit. The fact that each node stores the half of receiving flow can be considered as a physical way to reduce the flows and to terminate the process. So, this process will be terminated when there does not exist any flow to be send (see lines 8-18 of Algorithm 1).
  This process is based on Equation 1 in the sense that the nodes that are close to s will have high $S(x)$. In addition, it takes into account the graph structure in the sense that a node x that have a lot of connections with nodes of high stored flow, will receive also high quantity of flow. Therefore, when two nodes have the same distance from s, the node with more "important" connections[1] will have higher stored flow. For example, in Fig. 1 the nodes x and y have the same distance from s but $S(x) > S(y)$, due to the two connections of x with nodes a and b. So, the node x belongs on $C(s)$ with higher probability than node y.
- Based on $S(.)$, in the next step of the method the proposed method removes or add edges on s in order to remove bridges and to decrease the $d(x)$ for the nodes that belong on the $C(s)$. We sort the vector S in

[1]Connections with nodes that have high stored flow

descending order getting the node indices S_{ind} (see line 19 of Algorithm 1). If there exist a neighbor v of node s, that does not belong on the first $|n(s)|$ nodes of S_{ind} ($S_{ind}(1 : |n(s)|)$), then v is removed from the neighbors of s, and the stored as well as the transmitted quantity of v is transferred to node s, $T(s) = T(s) + T(v) + S(v)$. Since, with a high probability it holds that v does not belongs to the community of s (see lines 22-29 of Algorithm 1). Otherwise, we add the node $u = S_{ind}(|n(s)| + 1)$ to the node s neighbors (see lines 35-38 of Algorithm 1). In the next main step of the algorithm, we check if u is the last point of $S_{ind}(1 : |n(s)|)$ and then we we remove it for the neighbors of s (see lines 31-33 of Algorithm 1). The goal of the removal and the extension of neighbors of s is

  - to decrease shortest paths between the nodes that belong on the community of s and s in order to be able to increase their stored flow in the next iterations,
  - to gradually keep the most of the flow to the nodes of the community by removing bridges and
  - to keep the $|n(s)|$ balanced.

- The probability of a node x to belong to the community of s is equal or less than by the average of corresponding probabilities of node x neighbors (see Equation 2). So based on this inequality, in the case that $S(x)$ is greater than $E(S(n(x)))$, we set it to $E(S(n(x)))$. The quantity $S(x) - E(S(n(x)))$ is added to $T(s)$ (see lines 44-50 of Algorithm 1), so that in the next main step of the algorithm .
  This step will significant reduce the stored flow from nodes that does not belong on the community. For example, in Fig. 2, there exist a bridge (edge: $s \sim d$) and due to this step the reduction of $S(d)$ will be high. Since, the node d is only connected with almost fully connected subgraph G2. Without this step, $S(d)$ will be less but close to $S(a)$, $S(b)$ and $S(c)$.
- Finally, we sort the vector S in descending order and we compute the differences between adjacent elements of the sorted vector DS. Let K be the position of global minimum[2] of DS. The community of node s is defined by the first K nodes with highest $S(x)$. This trivial procedure is implemented by the function $getCluster$ (see line 51 of Algorithm 1).
- The main process ends when
  - the community finding algorithm converges to a solution (e.g. the last 10 iterations we receive the same community) or
  - the quantity $\frac{T(s)}{\sum_{x \in V} S(x)}$ is lower than a threshold meaning that $S(.)$ has been converged (see line 57 of Algorithm 1).

[2]In order to avoid some special - non real solutions and to speed up the algorithm we search for the global minima in the range $[|n(s)|, \frac{|\{x \in V : S(x) > 0\}|}{2}]$.

```
input : $V, W(i,j), i,j \in V, s \in V$.
output: $C$
1  $C = C_{prev} = \{s\}$
2  $T(x) = S(x) = 0, \forall x \in V$
3  $T(s) = |n(s)|$
4  $lastNode = doExit = iter = 0$
5  $W(x,s) = 0, \forall x \in V$
6  repeat
7      $iter = iter + 1$
8      repeat
9          $SET = \{x \in V : T(x) > 0.001\}$
10         foreach $x \in SET$ do
11             foreach $y \in n(x)$ do
12                 $ds = \frac{T(x)}{|n(x)|}$
13                 $S(y) = S(y) + 0.5 \cdot ds$
14                 $T(y) = T(y) + 0.5 \cdot ds$
15             end
16             $T(x) = 0$
17         end
18     until $SET = \emptyset$
19     $S_{ind} = sort(S)$
20     $SET = S_{ind}(1 : |n(s)|)$
21     $change = 0$
22     foreach $v \in n(s)$ do
23         if $v \notin SET$ then
24             $W(s,v) = 0$
25             $T(s) = T(s) + S(v) + T(v)$
26             $T(v) = V(v) = 0$
27             $change = 1$
28         end
29     end
30     $u = S_{ind}(|n(s)| + 1)$
31     if $S_{ind}(|n(s)|) = lastNode$ then
32         $W(s, lastNode) = 0$
33     end
34     $lastNode = 0$
35     if $change = 0$ then
36         $SET = SET \cup u$
37         $lastNode = u$
38     end
39     foreach $x \in SET$ do
40         if $x \notin n(s)$ then
41             $W(s,x) = 1$
42         end
43     end
44     $SET = \{x \in V : S(x) > E(S(n(x)))\}$
45     $S_{prev} = S$
46     foreach $x \in SET$ do
47         $DS = S(x) - E(S_{prev}(n(x)))$
48         $T(s) = T(s) + DS$
49         $S(x) = S_{prev}(x) - DS$
50     end
51     $C = getCluster(S) \cup \{s\}$
52     if $C = C_{prev}$ then
53         $doExit = doExit + 1$
54     else
55         $doExit = 0$
56     end
57 until $doExit \geq 10 \vee \frac{T(s)}{\sum_{x \in V} S(x)} < 0.02$
```

Algorithm 1: The proposed FlowPro algorithm.

IV. Experimental Results

We have created a variety of benchmark graphs with known community structure to test the accuracy of our algorithm. Our benchmark graphs were generated randomly given the following set of parameters: The number of nodes N of the graph (1000, 5000, 10000), the number of communities $Comm$ of the graph (5, 10, 20, 40, 80), the (average) degree of nodes $degree$ (10, 20, 30, 40) and the ratio of local links to node degree $local/degree$ (0.55, 0.65, 0.75, 0.85). In total, we created a number of 208 benchmark graphs. Notice that even though the number of the nodes, the number of the communities and the degree of the nodes are parameters of the construction of the graph, the degree of each node as well as the number of nodes in a single community varies based on a pareto distribution. This enables us to create graphs of community sizes and individual degrees varying up to an order of magnitude. The same dataset has been also used in [14].

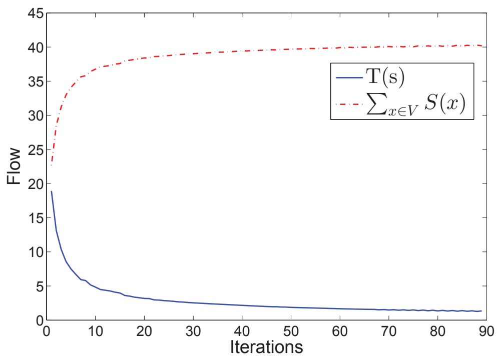


Fig. 3: An example of the variation of the stored flow and $T(s)$ during execution of the main process of FlowPro.

A demonstration of the propose method is given in [3], that contains the benchmark graphs, related articles and an executable of the proposed method.

We have used an accuracy-related metric found mostly used in the literature in order to measure the performance of community detection. The simple accuracy is defined as follows: Let A be the estimated and $\hat{A}$ the corresponding actual community. The accuracy (acc) is given by the fraction of the number of nodes that belong to the intersection of $A \cap \hat{A}$ divided by the number of nodes that belongs to the union $A \cup \hat{A}$.

$$acc = \frac{|A \cap \hat{A}|}{|A \cup \hat{A}|} \tag{3}$$

It holds that $acc \in [0, 1]$, the higher the accuracy the better the results. When $acc = 1$ the community detection algorithm gives perfect results.

Figs. 3 and 4 show an example of the evolution of the FlowPro for the synthetic graph with $N = 1000, Comm = 10$, $degree = 20$ and $local/degree = 0.75$. In Fig. 3 the variation of the stored flow ($\sum_{x\in V} S(x)$) and $T(s)$ during the execution of the main process is illustrated. It holds that $\sum_{x\in V} S(x)$) and $T(s)$ is increasing and decreasing, respectively, during the execution of the FlowPro. In Fig. 4 the FlowPro community detection result (nodes that are located before the black line) is depicted in the first, 31 and 61 iteration of the main process. According to the $getCluster$ procedure (see Section III), the nodes are sorted by their stored flow (S). The nodes that really belong on the community of s are plotted with blue spots while the rest nodes are plotted with red circles. In this example FlowPro is terminated since it converges to a solution. It holds that the acc of the first, 31 and 61 and 81 iteration is 23.2%, 94.3%, 97.1% and 100%, respectively.

[3]http://www.csd.uoc.gr/∼cpanag/DEMOS/commDetection.htm

Graph	FlowPro	Lancichinetti
ca-GrQc (Undirected, 5.242 nodes, 28.980 edges)	**0.264**	0.996
ca-HepTh (Undirected, 9.877 nodes, 51.971 edges)	**0.307**	0.999
ego-Facebook (Undirected, 747 nodes, 30.025 edges)	**0.135**	0.239
Wiki-Vote (Directed, 7.115 nodes, 103.689 Edges)	**0.678**	0.717

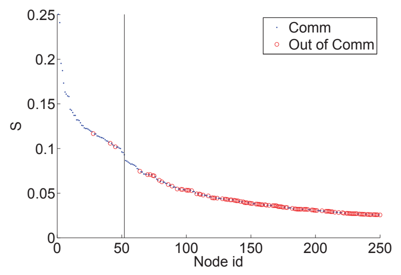
TABLE I: The Coverage of FlowPro and Lancichinetti on real world graphs.

The performance of FlowPro and the Lancichinetti [5] are compared on Benchmark graphs. Over all benchmark graphs, it holds that the mean accuracy of FlowPro and Lancichinetti is 81.7% and 34.1%, respectively. Hereafter, we have performed several experiments to show how the parameters of benchmark graphs affect the accuracy of algorithms.

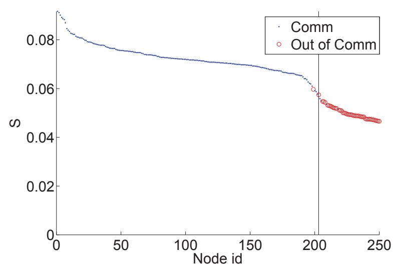
- If we ignore the cases of benchmark graphs that have N equal to 10000, then the mean accuracy of FlowPro and Lancichinetti is 82.1% and 35.1%, respectively.
- If we ignore the cases of benchmark graphs that have $degree$ equal to 10, then the mean accuracy of FlowPro and Lancichinetti is 85.5% and 38.4%, respectively.
- If we ignore the cases of benchmark graphs that have $Comm$ equal to 5, then the mean accuracy of FlowPro and Lancichinetti is 86.9% and 39.8%, respectively.
- If we ignore the cases of benchmark graphs that have $local/degree$ equal to 0.55, then the mean accuracy of FlowPro and Lancichinetti is 87.9% and 37.8%, respectively.
- If we ignore the cases of benchmark graphs that have $local/degree$ equal to 0.55 or $degree$ equal to 10 or $Comm$ equal to 5, then the mean accuracy of FlowPro and Lancichinetti is 94.2% and 48.5%, respectively.

According to these experiments it seems that the parameters $degree$, $Comm$ and $local/degree$ affect the performance of FlowPro and Lancichinetti since are related with the community density, while the size of the graph doesn't really affect the results of the FlowPro.

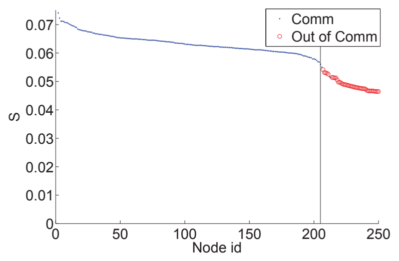
We also conducted experiments on four real world graphs of diverse sizes (see Table I). These graphs are obtained from [19]. The *conductance* metric [10] is used for comparison of results on real world graphs. These are different than those used in the case of benchmark graphs, since the real decomposition of graphs into communities is not known. We have tested each algorithm on 50 different nodes yielding the



(a) (1st iteration)



(b) (31 iteration)



(c) (61 iteration)

Fig. 4: The FlowPro community detection result in the first, 31 and 61 iteration of the main process.

average conductance. The *conductance* of a cut is a metric that compares the size of a cut and the number of edges in either of the two subgraphs induced by that cut. The lower conductance the better community detection. In Table I, we present the results of running FlowPro and Lancichinetti graphs using conductance metric. According to these results FlowPro clearly outperforms Lancichinetti, since a lower conductance value is better.

V. Conclusions and Future Work

We presented a local community finding algorithm which is based on a flow propagation method. The stored flow can be used as a belief of a node to belong on the community answering on several questions like find the k-nearest neighbors of the community of a node that are important on several applications of social networks. The novelty of the proposed approach is the fact that FlowPro is local, it detects a single community and it does not require the knowledge of the entire graph as most of the existing methods from literature. The proposed algorithm has been tested on a large number of benchmark graphs with known community structure comparing it with Lancichinetti algorithm [5], proving its effectiveness against another single community detection algorithm. We plan to expand the algorithm, in order to enable the detection of also non-overlapping and overlapping communities.

Acknowledgments

This research has been partially co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Programs: ARCHIMEDE III-TEI-Crete-P2PCOORD.

References

[1] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee, "Self-organization and identification of web communities," *IEEE Computer*, vol. 35, pp. 66–71, March 2002.

[2] D. Katsaros, G. Pallis, K. Stamos, A. Vakali, A. Sidiropoulos, and Y. Manolopoulos, "Cdns content outsourcing via generalized communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 137–151, 2009.

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

[4] S. Papadopoulos, A. Skusa, A. Vakali, Y. Kompatsiaris, and N. Wagner, "Bridge bounding: A local approach for efficient community discovery in complex networks," Tech. Rep. arXiv:0902.0871, Feb 2009.

[5] A. Lancichinetti, S. Fortunato, and J. Kertész, "Detecting the overlapping and hierarchical community structure in complex networks," *New Journal of Physics*, vol. 11, no. 3, pp. 033 015+, March 2009.

[6] A. Lancichinetti and S. Fortunato, "Community detection algorithms: a comparative analysis," *Physical Review E*, vol. 80, no. 5 Pt 2, p. 056117, Sep 2009.

[7] S. E. Schaeffer, "Graph clustering," *Computer Science Review*, vol. 1, no. 1, pp. 27 – 64, 2007.

[8] J. Chen and Y. Saad, "Dense subgraph extraction with application to community detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, pp. 1216–1230, 2012.

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

[10] H. Almeida, D. Guedes, W. Meira, and M. J. Zaki, "Is there a best quality metric for graph clusters?" in *Proceedings of the 2011 European conference on Machine learning and knowledge discovery in databases - Volume Part I*, pp. 44–59.

[11] H. Papadakis, C. Panagiotakis, and P. Fragopoulou, "Local community finding using synthetic coordinates," in *Future Information Technology*. Springer, 2011, pp. 9–15.

[12] H. Papadakis, P. Fragopoulou, and C. Panagiotakis, "Distributed community detection: Finding neighborhoods in a complex world using synthetic coordinates," in *ISCC'11*, 2011, pp. 1145–1150.

[13] H. Papadakis, C. Panagiotakis, and P. Fragopoulou, "Locating communities on real dataset graphs using synthetic coordinates," *Parallel Processing Letters*, vol. 22, no. 01, 2012.

[14] ——, "A distributed algorithm for community detection in large graphs," in *International Conference on Advances in Social Network Analysis and Mining*, 2013.

[15] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proceedings of the ACM SIGCOMM '04 Conference*, August 2004.

[16] C. Panagiotakis, H. Papadakis, E. Grinias, N. Komodakis, P. Fragopoulou, and G. Tziritas, "Interactive image segmentation based on synthetic graph coordinates," *Pattern Recognition*, vol. 46, no. 11, pp. 2940 – 2952, 2013.

[17] C. Panagiotakis and P. Fragopoulou, "Voting clustering and key points selection," in *International Conference on Computer Analysis of Images and Patterns*, 2013.

[18] B. J. Frey and D. Dueck, "Clustering by passing messages between data points," *science*, vol. 315, no. 5814, pp. 972–976, 2007.

[19] "Stanford large network dataset collection." [Online]. Available: http://snap.stanford.edu/data/