Fluid Communities: A Competitive and Highly Scalable Community Detection Algorithm

Ferran Parés*†, Dario Garcia-Gasulla*†, Armand Vilalta†, Jonatan Moreno†,
Eduard Ayguadé†‡, Jesus Labarta†‡, Ulises Cortés†‡ and Toyotaro Suzumura†§
†Barcelona Supercomputing Center (BSC)
‡Universitat Politècnica de Catalunya - BarcelonaTECH
§IBM T.J. Watson

***Abstract*—Community detection algorithms are a family of unsupervised graph mining algorithms which group vertices into clusters (*i.e.*, communities). These algorithms provide insight into both the structure of a network and the entities that compose it. In this paper we propose a novel community detection algorithm based on the simple idea of fluids interacting in an environment, expanding and contracting in contact with one another. The Fluid Communities algorithm is based on the propagation methodology, the most efficient approach to community detection in terms of computational cost and scalability. At the same time, the quality of the communities it finds is close to that of the current state-of-the-art community detection algorithms, and significantly superior to the Label Propagation Algorithm (LPA). While all previously proposed propagation-based algorithms can only produce a single clustering for a given graph, the Fluid Communities algorithm can identify a variable number of communities. As a result, the proposed algorithm represents a distinct and scalable tool for analyzing the topology of large scale graphs at multiple degrees of granularity.**

***Keywords*—Community Detection, Network Analysis, Graph Mining, Unsupervised Learning.**

I. Introduction

Community detection is one of the most popular graph mining tasks due to its ability to provide structural information of a network without supervision. Communities are typically defined by sets of vertices densely interconnected, and sparsely connected with the rest of the graph. Hence, finding communities within a graph helps unveil the internal organization of a graph, and can also be used to characterize the entities that compose it (*e.g.*, groups of people with shared interests, products with common properties, *etc.*).

One of the first and still most relevant community detection algorithms proposed in the literature is the Label Propagation Algorithm (LPA) [1]. Although other community detection algorithms have been shown to outperform it, LPA remains relevant due to its scalability (with linear computational complexity $\mathcal{O}$(E)) and yet competitive results [2]. Inspired by the efficiency of LPA and its propagation methodology, in this paper we propose a novel community detection algorithm which we call Fluid Communities (FluidC) algorithm. This algorithm tries to mimic the behaviour of several fluids (*i.e.*, communities) expanding and pushing one another in a shared, closed and non-homogeneous environment (*i.e.*, a graph), until an equilibrium state is found. One of the most relevant features of FluidC is that it can find any number of communities in a graph (*i.e.*, one may specify the number of communities FluidC must find) simply by initializing a different number of fluids in the environment. To the best of our knowledge, FluidC is the first propagation-based algorithm with this powerful property, which allows the algorithm to provide insights into the graph structure at different levels of granularity. Finally, FluidC avoids the generation of monster communities (a well known limitation of LPA [3]) through the consideration of fluid densities in an intuitive, non-parametric manner.

This paper is organized as follows: In §II, we review the related work. The FluidC algorithm and its properties are defined in §III. In §IV, we compare FluidC with top community detection algorithms in terms of clustering performance, while in §V we compare it in terms of computational cost and scalability. §VI is dedicated to the reproducibility of our work, providing details on our experimental setting and links to implementations of the algorithm on two different graph processing libraries. Finally, in §VII we present our conclusions given the previously reported results.

II. Related Work

Community detection became a popular problem at the beginning of the 21st century, when several algorithms were proposed in a short span of time. The most recent evaluation and comparison of community detection algorithms was made in [2], where the following eight algorithms were compared in terms of Normalized Mutual Information (NMI) and computing time:

- Edge Betweenness [4]
- Fast greedy [5]
- Infomap [6], [7]
- Label Propagation [1]
- Leading Eigenvector [8]
- Multilevel (*i.e.*, Louvain) [9]
- Spinglass [10]
- Walktrap [11]

NMI measures the dependence between two variables (in this case the predicted communities and the ground truth communities) taking into account both the quality and the quantity of the predicted communities. The performance of

*Both authors contributed equally to this work
{ferran.pares, dario.garcia}@bsc.es

the algorithms was measured on artificially generated graphs provided by the LFR benchmark [12], which defines a more realistic setting than the GN benchmark [13], including scale-free degree and cluster size distributions. The authors conclude that the Multilevel algorithm is the most competitive in terms of community detection quality.

A similar comparison of community detection algorithms was previously reported in [14]. In this work twelve algorithms are considered, some of them present in the study of [2] (Edge Betweenness, Fastgreedy, Multilevel and Infomap) and some not [15]–[22]. In this work, the algorithms were compared under the GN benchmark, the LFR benchmark, and on random graphs. In their summary, authors suggest using Infomap, Multilevel and the Multiresolution algorithm [22] when studying the community structure of a graph, for obtaining a set of algorithm-independent communities.

Results from both [2] and [14] show that the *fastest* algorithm of the eight is the well-known LPA algorithm, due to the efficiency and scalability of its propagation methodology. Unfortunately, LPA has severe limitations, like its tendency to create *monster communities* with size over 50% of the graph. To avoid these monster communities, a variant of LPA called LPA-δ was proposed in [3], which assigns an score to each label and incorporates hop attenuation. As a result, labels cannot spread more than a certain number of steps, which prevents the creation of monster communities. Additionally, LPA-δ also includes vertex preference based on degree, which coupled with hop attenuation provides more consistent communities than the ones produced by LPA. Another variant of LPA, called Diffusion and Propagation Algorithm (DPA), was proposed in [23]. DPA includes several improvements such as a dynamic hop attenuation evaluated at each iteration based on the proportion of vertices that change its label, and vertex preference based on the relative position of vertices within a community. Both LPA and its variants find fixed number of communities for any given graph, not allowing the number of communities to be specified as a parameter of the algorithm.

All the previously mentioned algorithms only use the network topology in order to determine its communities. In certain domains however, additional observable metadata may be available, information which may be used to complement the ground truth. This approach has been shown to be problematic [24], as the metadata may be irrelevant to the network structure, or it may capture an aspect of the network structure different from the ground truth. Algorithmic evaluation in this context becomes uncertain, as it includes an inherent bias. This problem is not present in artificially generated graphs, where the generative process is known, as well as the ground truth partitions. In our experiments we will focus on artificially generated graphs, and topology-based community detection algorithms.

III. Fluid Communities Algorithm

The Fluid Communities (FluidC) algorithm is a community detection algorithm based on the idea of introducing a number of fluids (*i.e.*, communities) within a non-homogeneous environment (*i.e.*, a non-complete graph), where fluids will expand and push each other influenced by the topology of the environment until a stable state is reached. A specific fluid community will conquer parts of the environment which have a favorable topology (*i.e.*, which are strongly connected with its vertices) while losing some parts to other fluid communities. Significantly, as a fluid community spreads through more vertices its density decreases, which reduces its strength to conquer and defend vertices. Unlike the hop attenuation of the LPA variants, densities in FluidC depend exclusively on the community size, and may increase or decrease regardless of the initialization setup.

Consider a graph $G = (V, E)$ formed by a set of vertices V and a set of edges E. FluidC initializes k Fluid Communities on k different random vertices of V, communities that will begin expanding throughout the graph. At all times, each fluid community has a total summed density of 1. When a fluid community is compacted into a single vertex (*e.g.*, at initialization), such vertex holds the full community density (*i.e.*, 1.0), which is also the maximum density a single vertex may hold. As a community spans through multiple vertices, its density becomes evenly distributed among the vertices that compose it (*e.g.*, if a community is composed by two vertices, each will hold 0.5 of the community's density).

The FluidC workflow follows the propagation approach introduced by LPA. On each superstep, FluidC iterates over all vertices in random order, updating the community each vertex belongs to using an update rule. Simply put, the update rule sums the densities of vertex neighbors, including itself, community-wise and returns the community with maximum density. This is equivalent to compute the total density per community of the vertex ego network. If the update rule provides two or more communities $\mathcal{C}_v^{'}$ with equal maximum density, but the current community of the vertex v is not among those, a random community is chosen among $\mathcal{C}_v^{'}$ as the new community of v. If the current community of v is among the set of communities with maximum density $\mathcal{C}_v^{'}$, v does not change its community. This rule guarantees that no community will ever be eliminated from the graph, since, when a community c is compressed into a single vertex v, c has the maximum possible density on the update rule of v (*i.e.*, 1.0). Formally, we define the updating rule as follows

$$\mathcal{C}_v^{'} = \underset{c \in \mathcal{C}}{argmax} \sum_{w \in \{v, \mathcal{N}_v\}} \mathcal{D}_w \cdot \delta(\mathcal{C}_w, c) \tag{1}$$

$$\delta(\mathcal{C}_w, c) = \begin{cases} 1, & if\ \mathcal{C}_w = c \\ 0, & if\ \mathcal{C}_w \neq c \end{cases} \tag{2}$$

where v is the vertex being updated, $\mathcal{C'}_v$ is the updated community of v, $\mathcal{N}_v$ are the neighbors of v, c refers to a community from the set of all communities $\mathcal{C}$, $\mathcal{D}_w$ is the density assigned to vertex w and $\mathcal{C}_w$ is the community vertex w belongs to. $\delta(\mathcal{C}_w, c)$ is the Kronecker delta between $\mathcal{C}_w$ and c community.

An example of the FluidC algorithm behavior is shown in Figure 1, and the complete pseudo-code of the algorithm is shown in the Appendix. FluidC is originally designed to be asynchronous, where each vertex update is computed using the latest partial state of the graph (some vertices may have updated their label in the current superstep and some may not). Notice that a straight-forward synchronous version of FluidC

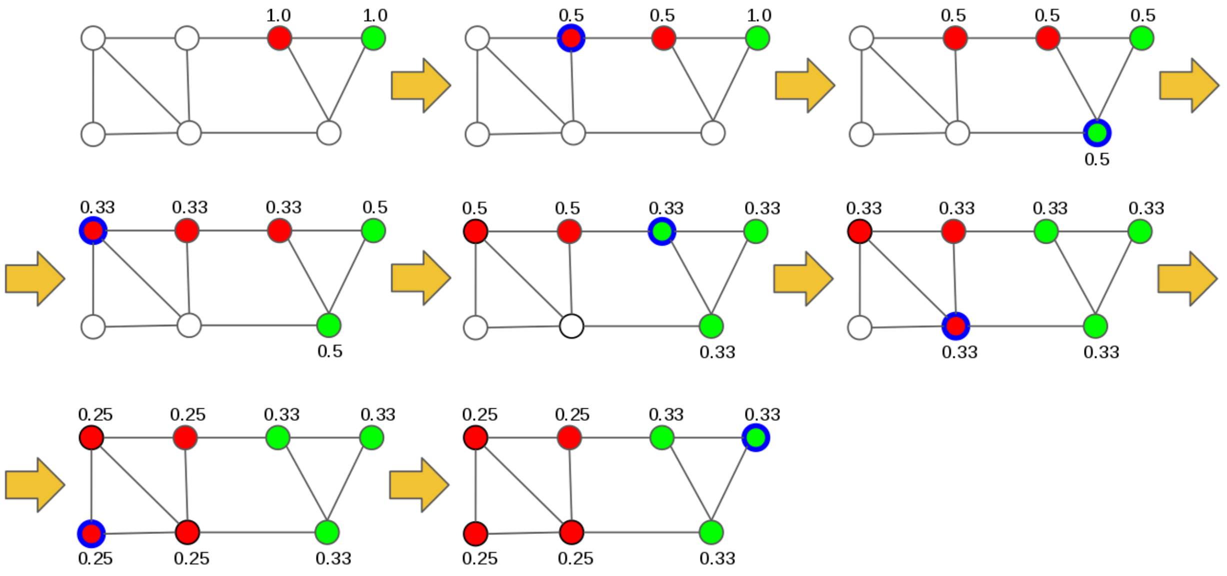

Fig. 1. FluidC algorithm workflow for k=2 communities (red and green). Each labeled vertex has its associated density. Label update rule is evaluated on each step for the highlighted vertex (blue). The algorithm converges after updating each vertex once (*i.e.*, one full superstep).

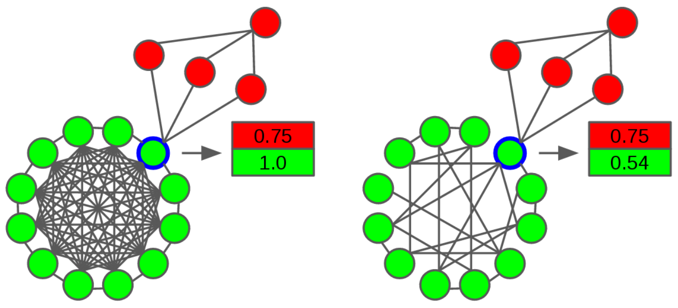

Fig. 2. Two cases of update rule on a vertex (highlighted in blue). Left case shows a densely connected green community which can successfully defend all its vertices. In the right case the green community is sparser and will lose the highlighted vertex (but only this one) to the red community.

(*i.e.*, one where all vertex update rules are computed using the final state of the previous superstep) would not guarantee that all communities have a total density of 1.0 at all times. Consequently, a community could lose all its vertices and be removed from the graph. FluidC is designed to process undirected, unweighted graphs, and variants of FluidC for directed and/or weighted graphs remain as future work.

Significantly, FluidC allows for the specification of the number of communities to be found, simply by initializing a variable number of fluids in the environment. This is a powerful and desirable property for data analytics, as it enables the study of the graph and its entities at several levels of granularity. Although a few community detection algorithms already had this feature (*e.g.*, Walktrap, Multilevel), none of those were based on the efficient propagation method.

All community detection algorithms implicitly include a definition of community. In the case of FluidC, the definition of community is based on edge density [25]. Edge density is used by fluid communities to maintain their vertices against external intrusion. Vertices without a defined community will eventually be assigned one community which is frequent among its neighbors. And finally, each vertex is reevaluated on each superstep which ensures that the final assignment of communities combines both edge density at vertex level (through neighbor edges) and edge density at community level (through fluid community density).

Another interesting feature of FluidC is that it avoids the creation of monster communities in a non-parametric manner. Due to the spread of density among the vertices that compose a community, a large community (when compared to the rest of communities in the graph) will only be able to keep its size and expand by having a favourable topology (*i.e.*, having lots of intra-community edges which make up for the larger spread of density). Figure 2 shows two cases of this behavior, one where a large community is able to defend against external attack, and one where it is not.

Although FluidC starts from a random initialization, its results are robust. Since fluid communities are highly mobile, a vertex may change its communities during the execution of the algorithm, even if the vertex was the initialization point of the community (see Figure 1 for an example). Regardless of where initial community vertices are placed, fluid communities will push one another towards positions coherent with the graph topology. The robustness of FluidC to random initialization is demonstrated in the evaluation of §IV. As shown in Figure 3 Panel g, FluidC obtains highly stable NMI results (*i.e.*, with small standard deviations) over 20 independent executions.

A particularity of the FluidC algorithm is that a given fluid community c_1 can become disconnected by the intrusion of a second community c_2 at some point during the algorithm execution. Indeed, if c_2 conquers vertices of c_1, it could happen that c_1 becomes split in two. When this happens, both disconnected parts of c_1 share the density of the community, but cannot contribute to each others defense. This puts them at a disadvantage, and makes it easier for other fluid communities to completely conquer one of the split parts of c_1. If, regardless

TABLE I. HYPERPARAMETERS OF LFR BENCHMARK

Parameter	Value
Number of vertices ($\|V\|$)	233 - 22186
Maximum degree	$0.1\|V\|$
Maximum community size	$0.1\|V\|$
Average degree	20
Degree distribution exponent	-2
Community size distribution exponent	-1
Mixing coefficient μ	0.03 - 0.75

of this disadvantage, a community c_1 is composed by disconnected components by the time the algorithm converges, it could be argued that each disconnected component represents a different community. For consistency, in our evaluation we consider each community returned by FluidC as a whole, regardless of its vertices being connected or not. This is the same approach followed by other algorithms which may generate disconnected communities (*e.g.*, Walktrap).

IV. Evaluation

The evaluation and comparison of unsupervised learning methods, such as community detection algorithms, is controversial due to the inherent lack of a universal ground truth. Furthermore, since the No Free Lunch (NFL) theorem [26] also applies to the community detection problem [24], one cannot claim that an algorithm A universally outperforms another algorithm B, as there are as many cases where algorithm A will outperform B as cases where algorithm B will outperform A. In our particular analysis, where we evaluate a subset of problems of special interest (defined by the LFR benchmark), we may find significant differences between the performance of algorithms. In this context, interpreting, recognizing or just simply reporting on which type of problems a community detection algorithm outperforms another supposes a relevant contribution to the state-of-the-art.

As previously mentioned, we evaluate the performance of several community detection algorithms in terms of NMI performance. Among all normalization variants of the Mutual Information metric we use the geometric normalization, defined as follows:

$$NMI(\mathcal{C},\mathcal{C}') = \frac{\sum_i^{|\mathcal{C}|}\sum_j^{|\mathcal{C}'|} P(i,j) log\left(\frac{P(i,j)}{P(i)P(j)}\right)}{\sqrt{P(i)log(P(i))P(j)log(P(j))}} \quad (3)$$

$$P(i) = \frac{|\mathcal{C}_i|}{|V|} \quad (4)$$

$$P(i,j) = \frac{|\mathcal{C}_i \cap \mathcal{C}_j|}{|V|} \quad (5)$$

where $\mathcal{C}$ refers to a set of clusters $\mathcal{C} = \{\mathcal{C}_1, ..., \mathcal{C}_k\}$, such that $k = |\mathcal{C}|$, $P(i)$ is the probability distribution, $P(i,j)$ is the joint probability distribution, $|\mathcal{C}_i|$ is the number of vertices in cluster i, and $|V|$ total number of vertices.

To evaluate performance we use the LFR benchmark [12], measuring the NMI obtained on a set of graphs with varying properties. This set of graphs comes from combining six graph sizes ($|V|$ = 233, 482, 1000, 3583, 8916 and 22186) and 25 different mixing parameter values (μ from 0.03 to 0.75). The mixing parameter is the average fraction of vertex edges which connect to vertices from other communities. Formally, it is defined as:

$$\mu = \frac{\sum_{v\in V} k_v^{ext}}{\sum_{v\in V} k_v} \quad (6)$$

To guarantee consistency, 20 different graphs were generated for each combination of graph size and mixing parameter. This results in a total of 3,000 graphs (6 graph sizes $\times$ 25 mixing parameter values $\times$ 20 executions). This is the same evaluation strategy used in [2]. Besides the graph size and mixing parameter, the LFR benchmark also requires a list of hyperparameters to generate a graph. For consistency, we have use the same ones defined in [2], shown in Table I.

Results are reported in Figure 3 showing the performance of six different community detection algorithms (Panels [a-f]), and the proposed FluidC (Panel g). Each panel of Figure 3 contains two plots, the bottom one shows performance in terms of NMI, while the top one shows the corresponding standard deviation (Std). Each plot line represents the results of an algorithm on a different graph size (see panel legend), which is obtained by averaging the results obtained on the corresponding 20 unique graphs for the various values of μ (shown on the horizontal axis). Additionally, each panel contains two reference lines, a vertical one to mark the 0.5 mixing parameter (μ) and a dotted horizontal line to mark the perfect NMI score (NMI = 1.0).

Before analyzing the results, let us discuss two different aspects of the evaluation which should be clarified. First, the FluidC algorithm is designed to run on connected graphs while the LFR benchmark may generate disjoint ones. And second, FluidC (among others, such as Multilevel and Walktrap) requires to specify the number of communities to be found, which is an unknown parameter k. In order to solve these issues, we use the following methodology:

1) If an artificially generated graph is composed by disconnected subgraphs, an independent execution of FluidC is computed on each of those separately. Communities found on the different subgraphs are appended to measure the overall NMI.
2) For each execution of FluidC, we consider the number of communities k within $\left\{1, ..., \sqrt{|V|}\right\}$, where $|V|$ is the number of vertices in the graph. We report the results obtained using the k which obtains the highest degree of modularity. This is analogous to what is done by other algorithms which require k (*e.g.*, Multilevel and Walktrap).

As shown in Figure 3, FluidC produces competitive results on the LFR benchmark, outperforming most of the top community detection algorithms (Panels [a-d]) and being competitive in terms of NMI to the best ones (Multilevel and Walktrap, Panels [e-f]). Although the NMI results of FluidC are slightly suboptimal for small μ values, its performance remains competitive in high μ values, where the detection of communities

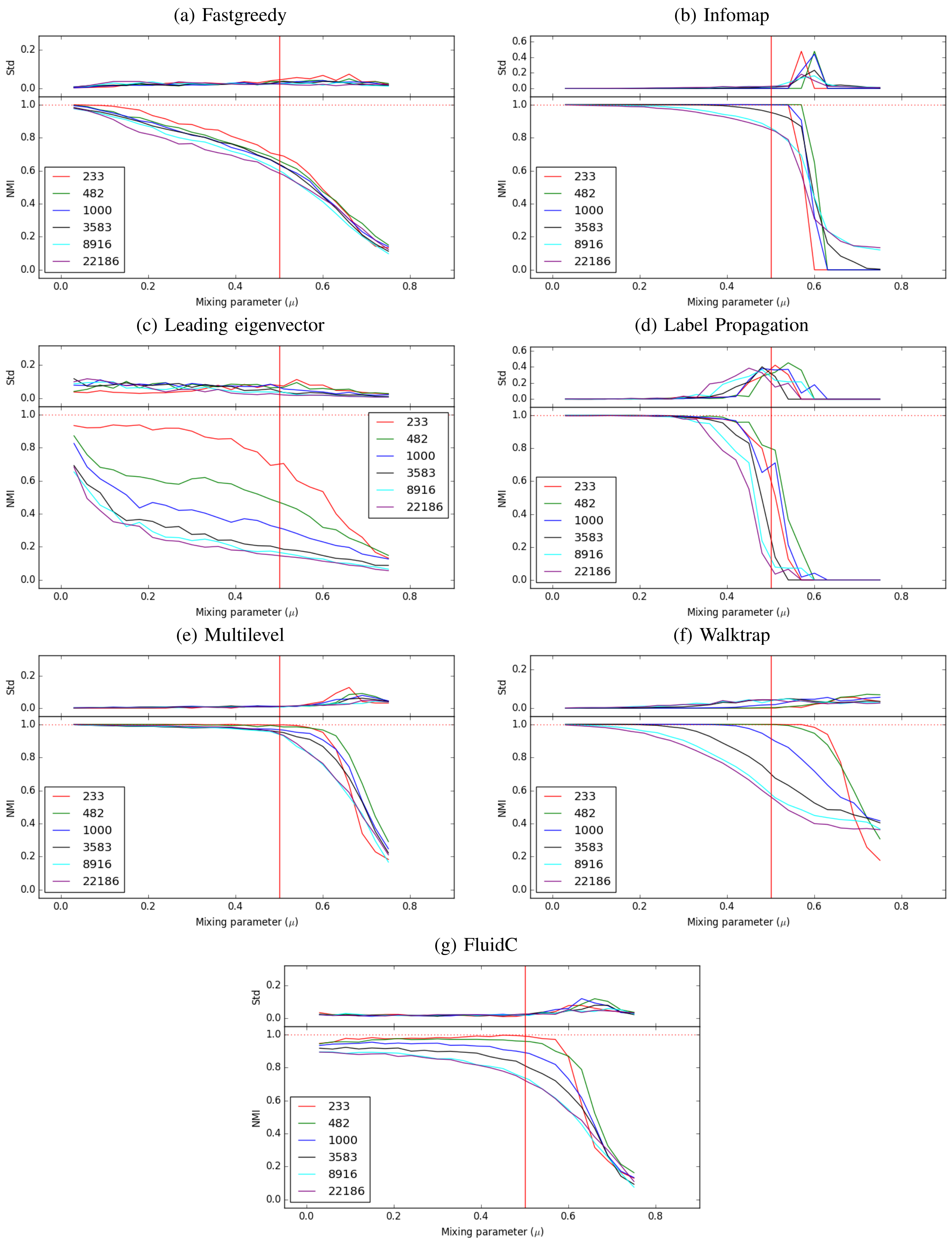

Fig. 3. Performance in NMI of six community detection algorithms and FluidC. Each Panel is divided in two plots, the bottom one shows the average NMI performance over 20 random graphs generated with the same properties (*i.e.*, size and mixing parameter), while the top one shows the standard deviation (Std). Different plotted lines correspond to different graph sizes. The Walktrap performance reported here differs significantly from [2]. This is due to an error in the experiments from Yang et. al., as identified by the same authors. After solving it, Yang et. al. were able to reproduce our results.

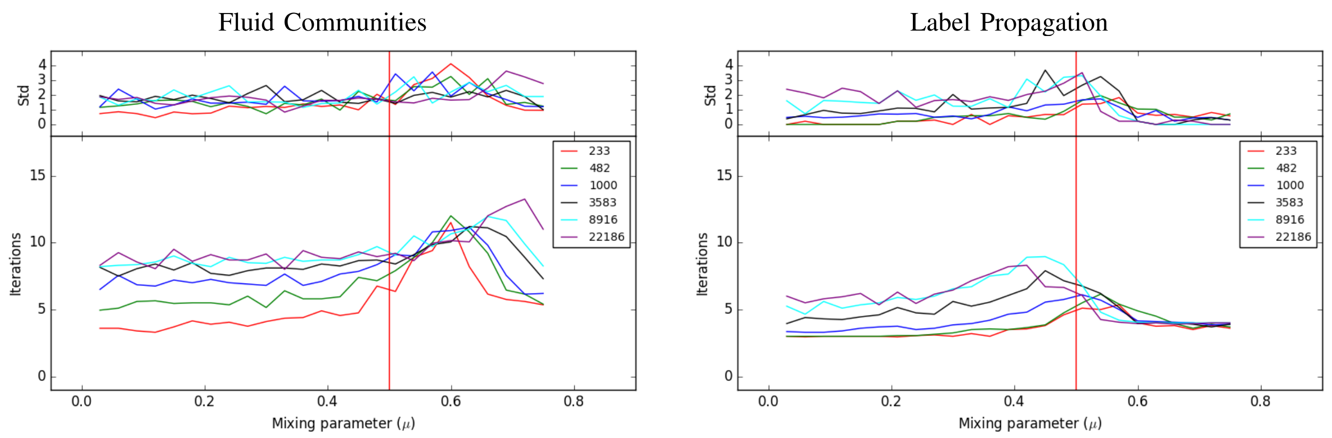

Fig. 4. Number of iterations until converge for the FluidC and LPA algorithms, when processing graphs of varying size and mixing parameter.

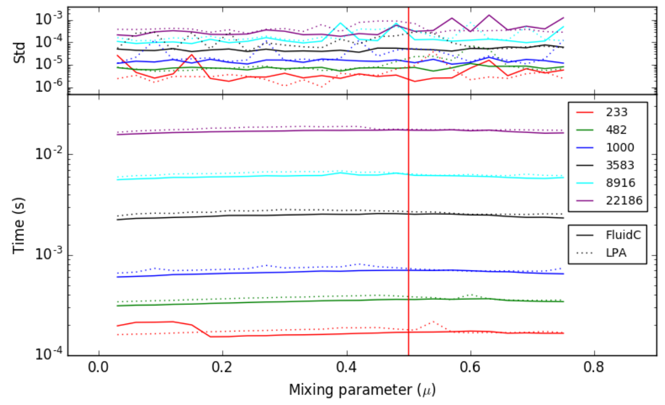

Fig. 5. Computing time per iteration for the FluidC and LPA algorithms, when processing graphs of varying size and mixing parameter.

is considerably more difficult and interesting. Significantly, FluidC outperforms LPA, its closest relative, for every graph size when the mixing parameter is above 0.4. FluidC also outperforms the competitive Walktrap algorithm for graphs with more than 1,000 vertices and mixing parameters above 0.4.

V. Scalability

The main purpose of the FluidC algorithm is to provide high quality communities in a scalable manner, so that good quality communities can also be obtained from large scale graphs. In the previous section we saw how the performance of FluidC in terms of NMI is very close to the best algorithms in the state-of-the-art (*e.g.*, Multilevel and Walktrap). Next we evaluate FluidC scalability, to show its relevance in the context of large networks.

To analyze the computational cost of FluidC we first compare the cost of one full superstep (checking and updating the communities of all vertices in the graph) with that of LPA. LPA is the fastest and more scalable algorithm in the state-of-the-art [2], which is why we use it as baseline for scalability along this section. Figure 5 shows the average time per iteration, using the same type of plots used in the NMI evaluation. Results indicate that the computing time per iteration of FluidC is virtually identical to that of LPA for all graph sizes and mixing parameters. Significantly, both algorithms are almost unaffected by a varying mixing parameter.

Beyond the cost of a single superstep, we also explore the total number of supersteps needed for the algorithm to converge. Figure 4 shows that information for both FluidC and LPA. For this experiment we set the FluidC parameter k to the ground truth. In the case of LPA, the number of iterations is rather stable for low mixing parameters ($\mu < 0.3$), and it slightly increases starting on μ values around 0.45. At $\mu > 0.6$ the number of iterations of LPA coverges to 4, probably because at this point LPA is unable to produce relevant communities (NMI = 0.0).

The number of iterations of FluidC can be divided in three phases based on the mixing parameter μ as follows:

1) When the mixing parameter is low ($\mu < 0.4$), the number of iterations is stable for each graph size, and slightly higher than LPA (less than twice as many iterations).
2) When the mixing parameter is close to $\mu = 0.6$, all graph sizes converge on 10 iterations. At this point FluidC is largely outperforming LPA in terms of NMI, only by doing a few more iterations.
3) When the mixing parameter is high ($\mu > 0.6$), the number of iterations becomes varied again, and is correlated with the graph size.

This results indicate that FluidC and LPA are similar both in time per iteration and number of iterations, which implies that both algorithms belong to the same family in terms of scalability. To provide further evidence in that regard, and to also evaluate the scalability of the most relevant alternatives, next we consider the evaluation of larger graphs. In detail we generate graphs with 60,000 and 150,000 vertices following the same methodology described in §IV, and measure the computing times of LPA, FluidC, Multilevel and Walktrap. Figure 6 shows the scalability of each algorithm, where the continuous lines in the background correspond to different mixing parameters (from 0.03 to 0.75), and the big dashed line with markers indicates the computed mean over all the 25 mixing parameters.

According to the results shown in Figure 6, LPA has the

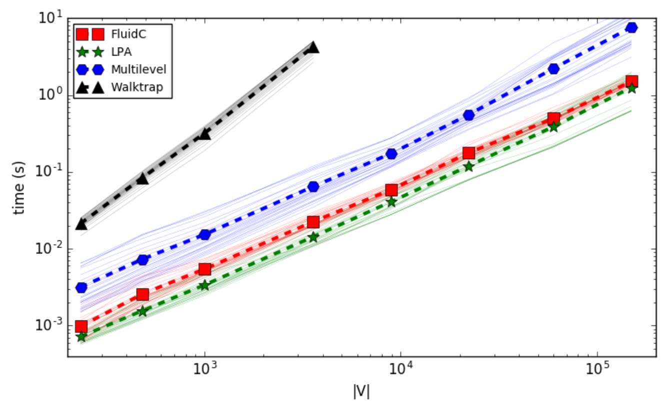

Fig. 6. Scalability of FluidC, LPA, Multilevel and Walktrap on graphs up to 150,000 vertices. Dashed line shows the average over the various mixing parameters values (from 0.03 to 0.75).

lowest computing time, closely followed by FluidC. However, LPA results are mistakenly optimistic, since the algorithm is particularly fast for large mixing parameters, where it obtains zero NMI after doing only three supersteps (see Panel d of Figure 3, and Figure 4). If those cases are not considered, LPA and FluidC have an analogous computing time and scalability.

Walktrap is considerably slower than the rest, and results for graphs larger than 3,000 vertices are not shown. Multilevel is roughly one order of magnitude slower than LPA/FluidC, and its cost grows faster. While the slope of LPA/FluidC computed through a linear regression is roughly 10^{-6}, the slope of Multilevel is close to 10^{-5}. The slope of Walktrap on the other hand is around 10^{-3}. According to [2], the computational cost and scalability of the rest of algorithms evaluated in Figure 3 is equal or worse than Walktrap.

VI. Reproducibility

All the experiments presented in this paper have been computed on the following environment:

- OpenSUSE Leap 42.2 OS (64-bits)
- Intel(R) Core(TM) i7-5600U CPU @ 2.60GHz
- 16GB DDR3 SDRAM

An open source implementation of the FluidC algorithm has been made available to the community at Github (github.com/FerranPares/Fluid-Communities). Furthermore, it is integrated into the graph libraries `networkx` (github.com/FerranPares/networkx) and `igraph` (github.com/FerranPares/igraph). For consistency, all scalability experiments were performed using the `igraph` graph library.

VII. Conclusions

In this paper we propose a novel community detection algorithm called Fluid Communities (FluidC). Through the well established LFR benchmark we demonstrate that FluidC identifies high quality communities (measured in NMI, see Figure 3), outperforming most algorithms, and getting close to the current best alternative in the state-of-the-art (*i.e.*, Multilevel). The main limitation of FluidC in terms of NMI performance is that it does not fully recover the ground truth communities on graphs with small mixing parameters. However, at larger mixing parameters (a more realistic environment) FluidC outperforms most of the alternatives, becoming the second best algorithm in larger graphs. Although FluidC does not clearly outperform the current state-of-the-art in terms of NMI, the importance of the contribution can be summarized both in terms of scalability and diversity.

In terms of scalability, FluidC together with LPA represents the state-of-the-art in community detection algorithms. Both belong to the fastest and most scalable family of algorithms in the literature, as shown in §V. However, while the performance of LPA rapidly degrades for large mixing parameters, FluidC is able to produce relevant communities for all mixing parameters. The next algorithm in terms of scalability is the Multilevel algorithm, which takes roughly one order of magnitude more seconds to compute, and which scales slightly worse (see Figure 6 and their mentioned slopes). Thus, we consider FluidC to be highly recommendable for computing graphs of arbitrary large size.

In terms of diversity, FluidC is the first propagation-based algorithm to report competitive results with the state-of-the-art (*i.e.*, Multilevel and Walktrap), and also the first propagation-based algorithm which can find a variable number of communities on a given graph. Providing coherent and diverse communities is particularly important for unsupervised learning tasks, such as community detection, where typically there is not a single correct answer. In this context, while algorithms like Multilevel can provide communities based on a modularity-based approach, FluidC can be used to generate a different set of communities from an edge density perspective.

Acknowledgements

This work is partially supported by the Joint Study Agreement no. W156463 under the IBM/BSC Deep Learning Center agreement, by the Spanish Government through Programa Severo Ochoa (SEV-2015-0493), by the Spanish Ministry of Science and Technology through TIN2015-65316-P project and by the Generalitat de Catalunya (contracts 2014-SGR-1051), and by the Core Research for Evolutional Science and Technology (CREST) program of Japan Science and Technology Agency (JST).

References

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

[2] Z. Yang, R. Algesheimer, and C. J. Tessone, "A comparative analysis of community detection algorithms on artificial networks," *Scientific Reports*, vol. 6, 2016.

[3] I. X. Leung, P. Hui, P. Lio, and J. Crowcroft, "Towards real-time community detection in large networks," *Physical Review E*, vol. 79, no. 6, p. 066107, 2009.

[4] 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.

[5] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[6] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[7] M. Rosvall, D. Axelsson, and C. T. Bergstrom, "The map equation," *The European Physical Journal Special Topics*, vol. 178, no. 1, pp. 13–23, 2009.

[8] M. E. Newman, "Finding community structure in networks using the eigenvectors of matrices," *Physical review E*, vol. 74, no. 3, p. 036104, 2006.

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[10] J. Reichardt and S. Bornholdt, "Statistical mechanics of community detection," *Physical Review E*, vol. 74, no. 1, p. 016110, 2006.

[11] P. Pons and M. Latapy, "Computing communities in large networks using random walks," in *International Symposium on Computer and Information Sciences*. Springer, 2005, pp. 284–293.

[12] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Physical review E*, vol. 78, no. 4, p. 046110, 2008.

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

[14] A. Lancichinetti and S. Fortunato, "Community detection algorithms: a comparative analysis," *Physical review E*, vol. 80, no. 5, p. 056117, 2009.

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

[16] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 9, pp. 2658–2663, 2004.

[17] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.

[18] S. Dongen, "Performance criteria for graph clustering and markov cluster experiments," 2000.

[19] M. Rosvall and C. T. Bergstrom, "An information-theoretic framework for resolving community structure in complex networks," *Proceedings of the National Academy of Sciences*, vol. 104, no. 18, pp. 7327–7331, 2007.

[20] L. Donetti and M. A. Munoz, "Detecting network communities: a new systematic and efficient algorithm," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2004, no. 10, p. P10012, 2004.

[21] M. E. Newman and E. A. Leicht, "Mixture models and exploratory analysis in networks," *Proceedings of the National Academy of Sciences*, vol. 104, no. 23, pp. 9564–9569, 2007.

[22] P. Ronhovde and Z. Nussinov, "Multiresolution community detection for megascale networks by information-based replica correlations," *Physical Review E*, vol. 80, no. 1, p. 016109, 2009.

[23] L. Šubelj and M. Bajec, "Unfolding communities in large complex networks: Combining defensive and offensive label propagation for core extraction," *Physical Review E*, vol. 83, no. 3, p. 036103, 2011.

[24] L. Peel, D. B. Larremore, and A. Clauset, "The ground truth about metadata and community detection in networks," *arXiv preprint arXiv:1608.05878*, 2016.

[25] R. K. Ronhovde, R. Peter, and Z. Nussinov, "An edge density definition of overlapping and weighted graph communities," *arXiv preprint arXiv:1301.3120*, 2013.

[26] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

APPENDIX
FLUID COMMUNITIES PSEUDO-CODE

```
Require: $G = (V, E),\ num\_communities\ k > 0,\ max\_iterations > 0$
 1: $V_{rand} \Leftarrow V$ in random order
 2: for $i \in \{0..k\}$ do
 3:     $v \Leftarrow V_{rand}[i]$
 4:     $\mathcal{C}_v \Leftarrow i$
 5:     $\mathcal{D}_v \Leftarrow 1.0$
 6: end for
 7: $converged \Leftarrow False$
 8: while $num\_iterations < max\_iterations$ and $converged = False$ do
 9:     $converged \Leftarrow True$
10:     for $v \in V$ in random order do
11:         $v\_new\_community \Leftarrow$ Community Update$(G, v)$
12:         if $v\_new\_community \neq \mathcal{C}_v$ then
13:             $v\_old\_community \Leftarrow \mathcal{C}_v$
14:             $\mathcal{C}_v \Leftarrow v\_new\_community$
15:             Density Update$(G, v\_old\_community)$
16:             Density Update$(G, v\_new\_community)$
17:             $converged \Leftarrow False$
18:         end if
19:     end for
20: end while
```

Fig. 7. Fluid Communities Algorithm

```
Require: $G, v$
 1: $community\_density[\mathcal{C}_v] \Leftarrow \mathcal{D}_v$
 2: for $w \in Neighbors(v)$ do
 3:     if $community\_density[\mathcal{C}_w]$ exists then
 4:         $community\_density[\mathcal{C}_w] \Leftarrow community\_density[\mathcal{C}_w] + \mathcal{D}_w$
 5:     else
 6:         $community\_density[\mathcal{C}_w] \Leftarrow \mathcal{D}_w$
 7:     end if
 8: end for
 9: $\mathcal{C}_v^{'} \Leftarrow \mathcal{C}_v$
10: $max\_density \Leftarrow max(community\_density)$
11: if $community\_density[\mathcal{C}_v] < max\_density$ then
12:     $\mathcal{C}_v^{'} \Leftarrow rand(community\_density = max\_density)$
13: end if
14: return $\mathcal{C}_v^{'}$
```

Fig. 8. Community Update method

```
Require: $G, community\_to\_update$
 1: $V_l \Leftarrow v \in V, \mathcal{C}_v = community\_to\_update$
 2: for $v \in V_l$ do
 3:     $\mathcal{D}_v \Leftarrow 1.0/|V_l|$
 4: end for
```

Fig. 9. Density Update method