Clustering Algorithm for Determining Community Structure in Large Networks

Josep M. Pujol,* Javier Béjar, and Jordi Delgado
Technical University of Catalonia, Software Department
Jordi Girona 1-3 A0-S106, 08034, Barcelona, Spain
(Dated: February 8, 2006)

We propose a novel algorithm to find the community structure in complex networks based on the combination of spectral analysis and modularity optimization. The clustering produced by our algorithm is as accurate as the best algorithms on the literature of modularity optimization, however, the main asset of the algorithm is its efficiency. The best match for our algorithm is Newman's fast algorithm, which is the reference algorithm for clustering in large networks due to its efficiency. When both algorithms are compared our algorithm outperforms the fast algorithm both in efficiency and accuracy of the clustering, in terms of modularity. Thus, the results suggest that the proposed algorithm is a good choice to analyze the community structure of medium and large networks in the range of tens and hundreds of thousand vertices.

1. INTRODUCTION

Clustering plays a key role in the analysis and exploration of data. In short, clustering is the method by which meaningful clusters, or groups, within collections of data are created. These clusters are intended to group individuals - or samples - who are similar to each other so that the hidden structure within the collection of data is revealed, resulting in a valuable acquisition of knowledge. Data-mining and machine learning are disciplines that extensively work with clustering, in particular, with datasets composed by individuals and attributes. The aim is to identify groups of individuals which are similar based on their attributes. However, thanks to the recent collective effort on analyzing and compiling very large networks, there is a growing interest in methods based on the structure - topology - of the networks rather than on the individuals' attributes.

This new method for clustering is possible thanks to the characterization of many systems as networks. Despite the very different nature of modelled systems (the Web [1], sexual relations [2], scientific collaboration[3, 4], protein interactions [5], the Internet [6], phone calls [7]) they do exhibit a non-trivial pattern of interactions. One of the regularities found in complex networks [8, 9] is the high cliquishness of the network [10], which leads to the fact that there are groups of vertices that are very interconnected among them with few interactions outside the groups. Therefore, there is an implicit community structure within complex networks.

Girvan and Newman [11] proposed an algorithm to extract the community structure from complex networks that has become one of the most used among the researchers in this community. From that important work a branch of research on complex networks has turned into clustering algorithms able to discover the community structure in those networks. To evaluate the accuracy - or quality - of a community structure yielded by a clustering algorithm Newman and Girvan devised a quantitative measure called *modularity* Q. Although there are other quantitative measures [12], modularity is widely accepted in the physics community. Q is defined in [13] as

$$Q = \sum_i \left(e_{ii} - a_i^2\right) \tag{1}$$

Modularity is the addition of the modularity of all the groups, $Q = \sum_i q_i$. Thus, for each group i that contains k vertices the modularity is calculated as the fraction of edges that have both ends pointing at vertices in group i, e_{ii}. The fraction of *intra-group* edges is confronted with the fraction of edges of that group, a_i, which are edges whose end points belong to at least one of vertices in i. This successful measure has been adopted not only to benchmark the accuracy of the clustering but also as the *fitness* value for clustering algorithms based on optimization. Finding the partition of groups that maximizes Q is believed to be a *NP-hard* problem, which makes a brute force exploration impossible for networks bigger than dozens of vertices. However, several search heuristics can be applied to explore the huge space of states to find a good partition. Following this approach, many algorithms have investigated different exploration heuristics to find the community structure while maximizing Q. Newman proposed in [14] a hill-climbing heuristic to create the hierarchy following an agglomerative strategy. The baseline is that every single node is a cluster, then the pair of clusters whose union produces the biggest increment in Q are merged into one. The process is repeated until only one cluster remains. By following the merging operations, the hierarchy that reveals the community structure is built. However, a *hill-climbing* heuristic cannot escape a sub-optimal maximum. Therefore, other search heuristics were devised. For instance, Guimerà and Amaral [15] proposed a simulated annealing

*Electronic address: jmpujol@lsi.upc.edu

approach. Duch and Arenas [16] proposed an algorithm based on Extremal Optimization. Both algorithms were able to extract the community structure more accurately in terms of modularity, although they were not as efficient as Newman's Fast algorithm [14]. Newman has very recently proposed another clustering algorithm [17] which outperforms the previous algorithms in both modularity and efficiency, although it is not as efficient as his previous fast algorithm [14]. Danon et al. [18] have also very recently presented a modification of Newman's Fast algorithm that while maitaining its computation efficiency yields more accurate partitions in terms of modularity.

Modularity optimization methods are neither the first nor the only ones to work on clustering in complex networks. In [11] Girvan and Newman reviewed classical hierarchical clustering algorithms on networks, showing that some classical distance measures were not well suited to work with complex networks. While the review done by Girvan and Newman in [11] was essentially correct, it overlooked two relevant areas that were already working in clustering of complex networks. Sociology was addressing clustering in Social Network Analysis [19]. On the other hand, Computer Science was also working in clustering of a particular instance of complex networks: the Web. Gibson et al. [20] and Kumar et al. [21] addressed clustering based on the analysis of the links between Web pages. A common tool used to address clustering on the Web is spectral analysis. However, this technique is applicable to any kind of network, for instance, newsgroups [22] and protein networks [23]. Spectral analysis has also been used in many other areas besides clustering. For instance, in work-load distribution between processors [24] and to find the relevant vertices of a network [25, 26]. Obviously, not all clustering is limited to spectral analysis. Flake [27] proposed an alternative approach based on minimum cut-trees over expanding networks that worked over the Web and could be applied to other kind of networks. However, we find particularly interesting clustering based on random walks [28, 29], which can be seen as a particular case of the spectral analysis. The underlying idea behind clustering using random walks is very intuitive: if a random walker starts in a given node, it will tend to visit more often vertices that belong to the same community of the initial node. So, provided that there is community structure, a random walker will spend most of the time stuck within the community it started from.

Our algorithm is a combination of spectral analysis and modularity optimization in order to achieve a good compromise between efficiency and accuracy of the cluster. Spectral analysis is used to reduce the number of initial vertices of the network: by means of a set of random walkers we create an initial partition of the network in a number of groups much smaller than the initial number of vertices. Consequently, the number of merge operations required to build up the hierarchy is reduced. Asymptotically our algorithm has a complexity $O(n^2)$, which is the same complexity of Newman's fast algorithm [14]. However, in terms of computational cost it is more efficient since the complexity can be decomposed as $O(ns) + O(s^2)$, where n is the number of vertices and s is the number of groups in the initial partition produced by the random walkers. Despite s being smaller than n it is not upper-bounded by a sub-linear function of n, so that the complexity remains $O(n^2)$. Yet, it is clearly more efficient and allow us to analyze very large networks in reasonable time while maintaining high quality clustering.

2. ALGORITHM

The proposed algorithm, henceforth PBD (after the initials of the authors), consists in an agglomerative hierarchical clustering where the initial groups are those produced by an initial partition of the network. The first step of the algorithm consists of a process of s random walkers traversing the network. The transition probability matrix M is defined as

$$M = (A + I)D^{-1} \tag{2}$$

Where I is the identity matrix and D is a diagonal matrix of the form $D_{ii} = 1 + \sum_j A_{ij}$. Thus, M_{ij} is the probability to go to node j from node i. The process carried out by the random walkers is defined by

$$G^{t+1} = M'G^t, \tag{3}$$

G^t is the matrix that contains the probability distribution of each random walker, G^t_{ij} is the probability that the random walker j is at node i at time t. Usually, the process is repeated iteratively until the stationary state is reached. However, we are interested in the transient state for all random walkers, consequently the process is repeated until we obtain G^T, where T is set to 3. Therefore, each random walker has done three jumps, which is the minimum number of hops to complete the shortest path to the origin point. Once the stochastic process is finished, each node i is classified into the group j which corresponds to the largest column at row i in G. Through this process, the initial n vertices are classified in approximately s groups and all the vertices of the same groups share that they were visited the most by the same random walker. Consequently, this means that they have a high degree of neighbours in common which implies a community. While this method is far from perfect, it allows us to drastically reduce the initial number of groups. The final number of groups might not correspond exactly to s since random walkers could preclude others. A random walker i is precluded when all

vertices by a random walker i are also visited more often by other random walkers, consequently the visited nodes are classified into others groups rather than the group started by i. Further, since the markov process is only iterated T times there is no guarantee that all vertices will be visited at least once, in this case an extra group with a single node is created.

The partition of the network heavily depends on which vertices are seeds - origins - of the random walkers. This problem is very related to classical clustering algorithms [30, 31] such as *k-means* [32]. How many seeds are required and where to place them is an open question [33]. We propose a straight forward heuristic that selects which vertices will be the seeds for the random walkers, i.e. to define G^0. Let R be the fraction of the most connected vertices chosen as seeds. If $k_i \geq z$ a random walker will start at node i, where the connectivity threshold z is defined as the maximum connectivity that makes the partition composed of the most connected nodes larger or equal than R, $\sum_{j=z}^{j\leq max(k)} p(k_j) \leq R$, where $p(k_j)$ is the fraction of vertices with connectivity k_j. The parameter R allows us to decide approximately the number of seeds, although the initial number depends on the structure of the network. For $R = 1$ there would be too many seeds for the algorithm to be efficient. On the contrary, $R \sim 0$ would be very efficient but the partition would be ill-constructed. In our experiment we set R to $\frac{1}{5}$ obtain good results for a wide variety of networks, as shown in Table I and II. Future work will look into different heuristics to choose the seeds, the quality-efficiency tradeoff of our algorithm is really dependant in this process, and other heuristics more elaborate might provide better results than our current straight-forward selection rule.

The complexity of finding the seeds for the random walkers is $O(n)$. The connectivity distribution and the connectivy threshold z can be computed in linear time respect to the number of vertices. In fact, the first approach we tried was to get the $n \cdot R$ most connected nodes, which would entail a sort operation with cost $O(nlogn)$. This option was discarded in favor of the connectivity threshold due to the extra cost which our algorithm intends to minimize.

The stochastic process defined in equation 3 has the iterative multiplication of two matrices, M and G, of dimension $n \times n$ and $n \times s$ respectively. However, thanks to the sparseness of both networks the cost can be reduced from $O(n^2 s)$ to $O(ms)$, where m is the number of edges. For each random walker j its probability distribution can be calculated in the worst case scenario with cost $O(m)$. Thus, the final cost can be considered $O(ns)$ because the number of edges scales with n in the limit of large n.

Once the initial partition is created, the algorithm builds an agglomerative hierarchical clustering. This method consists in creating a series of partitions of the data: $C_s, C_{s-1}, ..., C_1$, where first C_s consist of s single clusters (groups), and the last C_1, consists of a single group containing all the individuals. The method iteratively joins the two individuals or clusters (groups of individuals) which are most similar. Thus, after $s - 1$ join operations the clustering is complete, and the result is a binary tree known as *dendrogram* that reveals the underlying structure of the data.

Let us say that the initial partition yielded s groups, despite the fact that it is a upper bound since some groups might be empty because their random walkers were precluded by others. For each group j, the contribution to the total modularity, that is $q_j = e_{jj} - aj^2$, can be calculated in linear time $O(s)$. The group that contributes the least to the total modularity Q - let us say j such that $j = argmin_k(q_k)$ - is selected to be joined to the group that maximizes the increment of modularity as defined in the following equation,

$$\triangle Q = (2e_{ij} + e_{ii} + e_{jj}) - (a_i + a_j)^2 - (e_{ii} - a_i^2) - (e_{jj} - a_j^2) \quad (4)$$

The increment in total modularity is the modularity of the merged group $(2e_{ij} + e_{ii} + e_{jj}) - (a_i + a_j)^2$ minus the contribution to the modularity of both groups; q_i and q_j. Equation 4 can be reduced trivially to Eq. 2 of Newman's Fast Algorithm [14],

$$\triangle Q = 2e_{ij} - 2a_i a_j = 2(e_{ij} - a_i a_j) \quad (5)$$

In the event that two candidates, i_1 and i_2, have the same effect over the total modularity, the candidate group chosen will be the one that has the least modularity, $min(q_{i_1}, q_{i_2})$. Thus, groups with low modularity are preferred in the merge operation.

The merge operation can be performed in the worst case scenario in linear time with respect to the current number of groups, thus $O(s)$. Furthermore, the operation needs to be done $s - 1$ times. Therefore, the complexity of building up the hierarchy is $O(s^2)$. The search heuristic proposed is extremely *greedy* since it only takes into consideration pairs of groups, provided that one groups is fixed. Conversely, Newman's fast algorithm calculates the gain of modularity for each possible pair of groups. Besides that, other algorithms based on modularity optimization usually have even more expensive search heuristics that allow a better exploration at the expense of efficiency. Our proposal was designed to focus on efficiency, as it can be seen in the heuristic decisions made by the algorithm. However, as we will show in the experiments section, this focus on efficiency does not necessarily imply a loss of quality of the clustering.

2.1. Parallelization

To reduce even further the execution time of the algorithm its parallelization could be easily implemented.

The stochastic process as defined in equation 3 can be carried out in parallel by different computers or processors. To calculate the probability distribution of a given set of random walkers, only the transition probability matrix is required. Thus, the matrix G of dimension $n \times s$ could be split column by column into a set of smaller matrices of dimension $n \times \varsigma$, where $\varsigma \ll s$. This would drastically reduce the cost of the stochastic process. Unfortunately, the modularity optimization step cannot be parallelized so easily. Thus, trivial parallelization would only affect the spectral analysis part of the algorithm. Although this part is the most expensive part in the algorithm $O(ns)$, the modurality optimization $O(s^2)$ would still be executed sequentially. Therefore, the asymptotic cost of the PBD algorithm would still be $O(n^2)$.

2.2. A Working Example with Zachary's Network

To illustrate our algorithm we include an execution on the Zachary network [34], which is a well-known dataset in the literature of community extraction. In figure 1 we can find the network at the different stages of the execution of the algorithm. Figure 1.a shows which vertices, labelled 2, are seeds of the random walkers, 16 in total. Thus, s is 16 compared to the original 34 vertices. In table II the relation between network size n and the number of random walkers s for a wide range of networks is shown. Figure 1.b shows the initial partition of the networks produced by the random walker process, described in equation 3. The initial 34 vertices are grouped into 13 groups. This partition has a modularity Q of 0.1547. From that point on the algorithm starts the modularity optimization stage governed by equation 4. At each step, two of the remaining groups are joined according to equation 4. Figures 1.c and 1.d show the network divided into 4 and 2 groups respectively.

The community structure of Zachary's is better seen in figure 2. The maximum modularity is obtained by the partition in 4 communities, achieving Q=0.3937. However, the original empirical work on the Zachary Karate Club [34] found two communities: those aligned with the instructor and those aligned with the administrator. The division into two groups produced by PBD algorithm produces a high modularity Q=0.3718. Also, the two original communities found empirically correspond to the communities found by the algorithm with the exception of node 10, which is misclassified.

It is evident that figure 2 is not a dendrogram over all the vertices of the network, but a dendrogram over the initial communities. As a consequence of the random walker process, the structure between vertices belonging to the same initial community remains unknown. However, this loss is negligible since the relevant high-level structure is not affected as can be seen in figure 2.

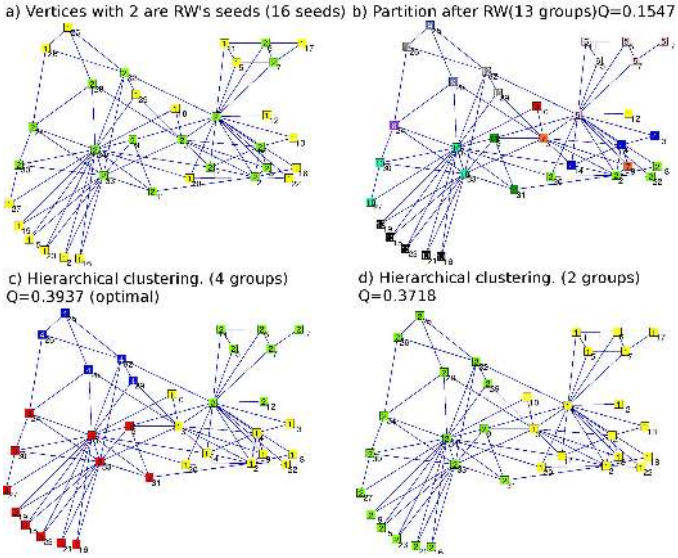

FIG. 1: (Color online) Working of the PDB algorithm in the Zachary network. Sub-figure *a* shows the initial vertices (labelled with 2) which are seeds for the random walkers vertices labeled. Sub-figure *b* shows the initial partition of the network into communities produced by the random walker stage. Sub-figures *c* and *d* show different partitions created by the modularity optimization process. The optimal partition, whose modularity is maximal, is shown in sub-figure *c*.

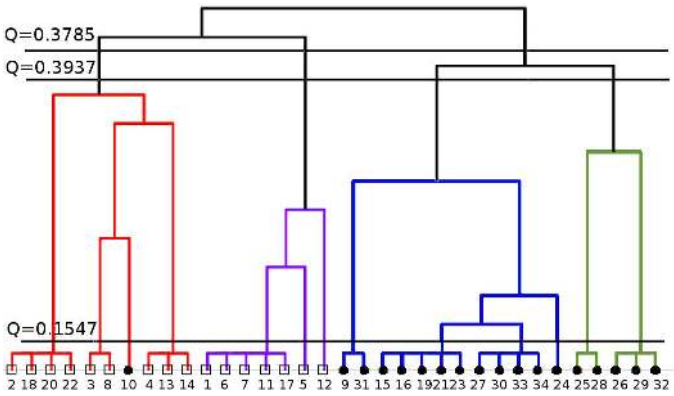

FIG. 2: (Color online) Community structure of Zachary network produced by PDB algorithm. Circles and squares over the individuals denote to who they align to after the karate club broke up. Those who aligned with the instructor are represented by circles, and those who aligned with the administrator are represented by squares.

3. EXPERIMENTS

In order to further analyze our algorithm we chose a set of ten different networks of different sizes, ranging from 34 to 498925 vertices. The networks modeled a wide spectrum of systems. There are social networks, such as the Zachary Karate Club [34] and the social network of the Software Department (LSI) at the Technical University of Catalonia [35]. Scientific collaboration networks such as

Network	*Size (n)*	Q_N	g_N	Q_{EO}	g_{EO}	Q_{PBD}	g_{PBD}
Zachary	34	0.3807	3	0.4176	4	0.3937	4
LSI	139	0.6428	6	0.6572	7	0.6604	6
C. Elegans	453	0.40	10	0.4376	10	0.4164	7
Directors Board	598	0.8046	21	0.8113	27	0.8273	16
Scientometrics	2678	0.5555	24	0.6042	19	0.5629	10
Erdös (2002)	6927	0.6723	57	0.6520	88	0.6817	20
Cond-Mat	27519	0.6653	324	0.6790	647	0.7251	44
Word-Net	75606	0.7963	453	N/A	N/A	0.7885	47
WWW ND	325729	0.9273	2192	N/A	N/A	0.9272	83
Actors ND	498925	0.7243	2113	N/A	N/A	0.7297	14

TABLE I: Comparison between maximum modularity Q and number of communities g obtained by Newman's fast algorithm (N), Duch and Arenas Extremal Optimization algorithm (EO) and PBD algorithm.

Cond-mat [4] and the Erdös collaboration network [40]. Citation networks such as Scientometrics [41]. Affiliation network among Spanish top director boards [42]. Network of relations between words such as WordNet [43]. Metabolic networks such as the *C. Elegans* [36]. A portion of the Web from the Notre Dame University dataset [1]. Finally, the last type of network was the movie collaboration [44] network, again obtained from the Notre Dame University dataset [37]. In all the networks we only worked with the biggest connex component, removing all multiple relations and self-reference edges.

Comparing Modularity

Table I summarizes the highest modularity achieved by Newman's Fast algorithm (Q_N) [14] and our algorithm (Q_{PBD}). For eight out of the ten tested networks the PBD algorithm produces a higher modularity, and the maximum difference in favour of the PBD algorithm is in the cond-mat network. Thus, in general the PBD algorithm yields a slightly better modularity than the Newman's Fast algorithm. However, as mentioned in the introduction, there exist in the literature other algorithms based on modularity optimization that also outperform the Newman Fast algorithm [14]. In table I we also included the results obtained by the extremal optimization algorithm (EO) by Duch-Arenas [16]. In this case the maximum modularity obtained by EO outperforms in two of three cases the modularity obtained using PDB, and in all the available cases it outperforms the modularity obtained using Newman's Fast Algorithm. However, the complexity of the algorithms that use elaborated search heuristics is superior to the complexity of both Newman's Fast algorithm and PDB algorithm. For instance, EO's complexity is $O(n^2 log^2(n))$. Thus, we can conclude that PDB has a good balance between efficiency and quality.

Comparing the Number of Communities

Clustering, though, does not only depend on obtaining the partition that maximizes modularity. The number of communities - or groups - contained in the optimal partition is also very important. In table I the number of communities is denoted by g_N, g_{EO}, g_{PBD}. We can clearly observe an order $g_{EO} > g_N > g_{PBD}$. Provided that maximum modularity does not greatly differ between partitions, the huge differences in the number of communities obtained by the different algorithms are indeed striking and requires further study.

Provided we assume that modularity Q is a good measure for community structure, we must take for granted that two partitions with similar modularity are equally accurate. Thinking otherwise would lead us to the conclusion that modularity is not a good measure. Finding a *bogus* partition that yielded higher modularity than a *good* partition would mean that modularity is not representative of the structure. Consequently, it could not be used as the fitness variable for the optimization. However we do not believe that this is the case.

So, if we assume that modularity is a good measure, what happens when two partitions having very similar modularity have a very different number of groups? A first approach would be to think that the partition with smaller number of groups is more general than the partition with the larger number of groups. Thus, partitions with smaller number of groups would be, in principle, more interesting for several reasons. (i) They would provide a more general perspective on the underlying structure of the network, since they would be able to find a meaningful partition at a higher level of the structure. (ii) A small number of groups would simplify the analysis of the obtained results. And (iii), general or *high-order* partitions could always be re-clustered to further analyze the structure of a particular group if a more detailed - fine-grained - analysis were required. The opposite could not be done, otherwise the algorithm would have detected the more general partition with higher modularity.

Table I shows that PBD algorithm yields more general partitions while having similar or better modularity. This effect is specially acute in large networks. One might conclude from the results of the experiments that both EO and N algorithms undergo an unnecessary overspecialization. For instance, let us we take the optimal partition of the cond-mat network. PBD provides the highest modularity and the smallest number of partitions, which is 647, 324 and 44 using EO, N and PBD algorithms respectively.

Figure 3 shows the evolution of Q in the cond-mat network using N and PDB algorithm. Notice that the modularity obtained by the N algorithm is very close to its maximum modularity when the number of remaing groups is in the range between 40 to 1000. Once the par-

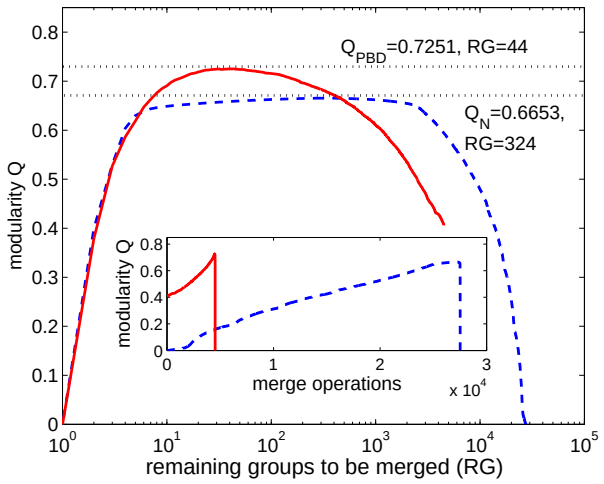

FIG. 3: Modularity Q over the execution of the algorithms for the cond-mat network. PBD algorithm is shown as the solid line and Newman's Fast algorithm (N) as the dashed line. Notice that the number of merge operations required for both algorithms is different (PBD optimization stage starts with the partition given by the spectral analysis stage). PBD algorithm requires aproximately the 22% of the merge operations performed by the N algorithm. Both the figure and the inset contain the very same information but it is depicted in a different fashion to better illustrate the evolution of modularity. Notice that the main figure is plotted in a log scale and the time goes from right to left. This is done in order to magnify the last steps of the algorithms that is when maximum modularity is found. We would like to remark that the number of remaing groups decreases over time, while the number of merge operations increases over time.

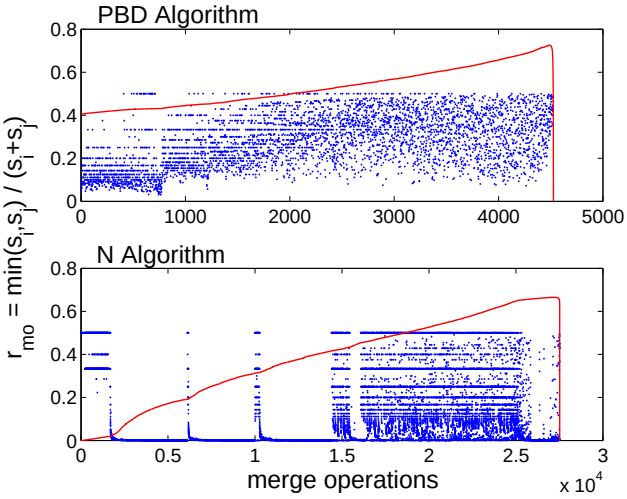

FIG. 4: Behaviour of the merge operation of the PBD (above) and the N (below) algorithms. For each merge operation the normalized ratio of the size of groups to be merged is calculated as $r_{mo} = \frac{min(s_i,sj)}{s_i+s_j}$, where s_i and s_j are the size of group i and j respectively. The merge operation creates a new group $z = i \cup j$ of size $s_z = s_i + s_j$. The line corresponds to the evolution of modularity Q.

tition contains more than 1000 groups it does not increase or decrease substantially until the number of groups in the partition reaches 40, at that point, modularity starts to decrease abruptly. This range of stable modularity contains a partition with 44 groups which is the maximum modularity found by PBD algorithm. This behaviour is not observed in the evolution of modularity for the PBD algorithm, which keeps increasing modularity until it gets to the maximum and then it starts a quick descent. Being in a *plateau* where modularity is neither increasing or decreasing significatively might suggest that the search heuristic of N algorithm could be stuck in a local sub-optimal state.

A more detailed study of the internal behaviour of N and PBD algorithm is described in figure 4, where the evolution of the normalized size of groups to be merged is depicted. Each merge operation joins group i and j in a new group z. The size of the two groups chosen by the algorithm to be merged is expressed as $r_{mo} = \frac{min(s_i,sj)}{s_i+s_j}$. The ratio r_{mo} has values in the range $(0..\frac{1}{2}]$, when r_{mo} is close to zero means that there is one group that is clearly bigger than its counterpart. When $r_{mo} = \frac{1}{2}$ both groups have the same size. It is easy to see how the N algorithm tends to join groups of very different size, whereas the PBD algorithm tends to do the opposite, i.e. it prefers groups of similar size. Consequently, the N algorithm produces at the very beginning groups of extremly large size by joining a single vertice to a very large group, when this group cannot accept more vertices a new group is started. This behaviour is clearly observed at operation 6000, 10000 and 15000 approximately, creating three massive groups of 13000 vertices and leaving the rest of groups composed mostly by individual vertices. This might be the reason why the search heuristic fails to increase modularity in the *plateau* of figure 3. Approximately from operation 16000 to 26000 this behaviour is not so evident, where groups of similar same size are often merged. However, the behaviour that merges very dissimilar groups appears again between operations 26000 and 27000, overlapping the *plateau* of modularity. At this point, big groups cannot be merged obtaining a gain of modularity, therefore, the only possible option left to be explored by the search heuristic is to merge *left-over* groups of very few vertices into the existing big groups. As a consequence, the search heuristic gets stuck in a situation where changes of modularity are minimum and there is no escape from the sub-optimal partition. The bias of the N algorithm towards creating very large groups has also been very recently reported by Danon et al. in [18]. We can see how this behaviour is not present in the PBD algorithm. The greedy search heuristic used by PBD algorithm forces the worst group to be the one going to be merged, and this results in groups

having similar sizes. By doing so, the search heuristic has many possible combinations to explore, avoiding getting trapped too early in a local sub-optimum as it happened in the N algorithm. That is the reason why the PBD algorithm is able to find in the cond-mat network a higher modularity partition which also has less groups. As for the EO algorithm, we could not carry out the same analysis but it is plausible that the resulting large number of groups can be attributted to its divisive clustering strategy.

In order to analyze with more detail the partitions yielded by the N and the PDB algorithms the distance between the optimal partitions for the cond-mat network are calculated. Gustafsson et al [38] reviewed different distance measures to compare partitions of the same network. Since we want to look into the hypothesis that the partition yielded by PDB (P_{PBD}) is more general than the partition yielded by N algorithm (P_N) we chose the m_{div} measure, which is the minimum number of divisions to be applied to partitions A and B to obtain the partition C defined in [39] as,

$$C = \bigcup_{i=1}^{|A|} \bigcup_{j=1}^{|B|} (a_i \cap b_j) \tag{6}$$

Partition C is the union of all possible intersections between the groups in A and B. We rename C as P_{N-PBD}. The distance between partition P_{PDB} and P_{N-PBD} is 894, which is the number of divisions to be applied to P_{PBD} in order to obtain P_{N-PBD}. The distance between P_N and P_{N-PBD} is 614. Thus, the total distance is 1508, which is the sum of both distances. In order to have a baseline comparison for the distance between P_{PBD} and P_N we created a random partition P_{rand} with the same cardinality as P_{PBD}. The random partition was replicated 30 times and so was the measurement, the average distance between P_N and P_{rand} was 7166.7 with a standard deviation of 33.65.

Comparing Efficiency

After comparing the quality of the clustering produced by the PBD algorithm we must turn our attention towards its performance. As we already mentioned, all design decisions were biased towards improving the efficiency so that the algorithm could cope with medium and large networks, which other algorithms cannot handle in reasonable time. Table II summarizes the runtime (cpu-time) of our algorithm compared to Newman's Fast algorithm, which is the reference algorithm due to its efficiency. We are perfectly aware of the problems related to comparison of algorithms based on run-time instead of only considering their complexity. In order to allow a fair comparison between both algorithms we implemented them from scratch optimizing them to the best of our abilities. Needless to say the runs were executed on the same desktop computer (Pentium 4, 3Ghz) exclusively dedicated to the experiment. Table II shows that PBD is much faster than Newman's Fast algorithm for networks bigger than a thousand vertices. Conversely, the PBD algorithm is slower than the N algorithm for small networks. This is due to the two sequential processes - spectral analysis and modularity optimization - that take place in the PBD algorithm. The difference in execution time heavily depends on the number of ran-

Network	*Size (n)*	t_N	t_{PBD}	s_{PBD}
Zachary	34	0.002	0.014	16
LSI	139	0.003	0.015	42
C. Elegans	453	0.026	0.064	118
Directors Board	598	0.038	0.031	125
Scientometrics	2678	1.6	0.320	619
Erdös (2002)	6927	3.14	2.6	2155
Cond-Mat	27519	125.8	11.2	6224
Word-Net	75606	490.6	204.1	38701
WWW ND	325729	10932.1	1775.6	86908
Actors ND	498925	34208.3	3326.3	118897

TABLE II: Comparison between CPU-time t (in seconds) between Newman's Fast algorithm and the PBD algorithm. It also includes the number of random walkers required to create the initial partition s_{PBD}

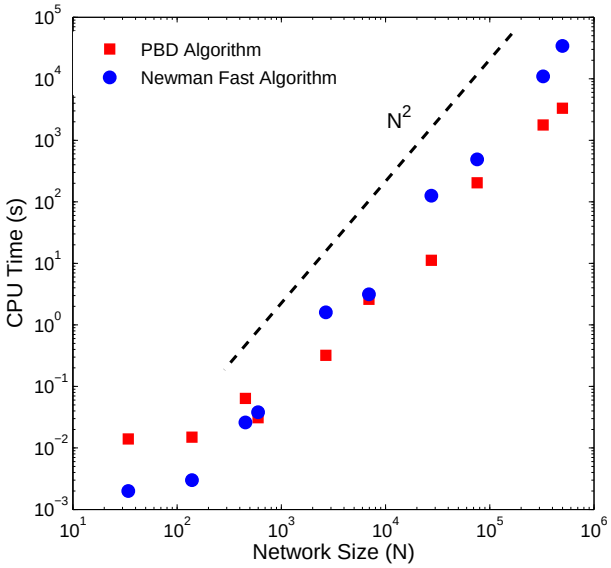

FIG. 5: Comparison between the efficiency in CPU-time in seconds for 10 networks of different sizes. The solid circles show the results for the Newman's Fast algorithm. The results of the PBD algorithm are represented by solid squares.

dom walkers (s_{PDB}) required for the first stage of PDB algorithm. The smaller the ratio between network size and the number of random walkers, the faster the PDB algorithm is.

Figure 5 shows graphically the relation between network size and the running time of both algorithms already seen in table II. The PDB algorithm clearly outperforms Newman's Fast algorithm, often by one order of magnitude. However, the asymptotic quadratic behaviour of both algorithms is evident, which lead us to think that our algorithm cannot scale to very large networks of millions of nodes. Resorting to parallelization would allow us to analyze networks of a few million nodes in an acceptable time, however, PDB will undoubtedly become too slow for very large networks. Nonetheless, it allows to shift the network size threshold far enough to be useful for medium and large networks. Reduction of approximately one order of magnitude allows to shift from minutes to seconds, or from hours to minutes. For instance, the clustering of the largest network we had access to was reduced from approximately 9 hours to just one.

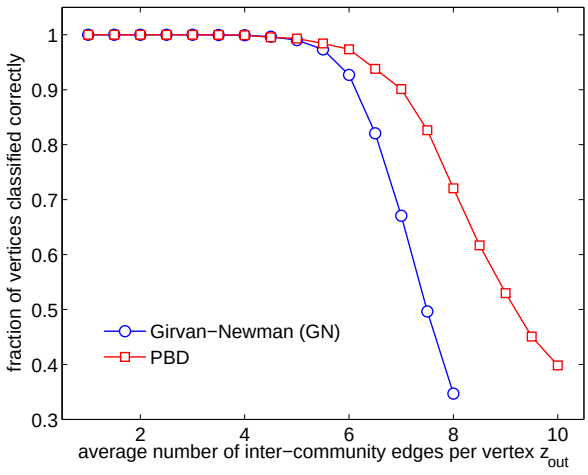

FIG. 6: Fraction of vertices correctly classified using computer-generated networks. The circles show the result of the Girvan-Newman algorithm (GN). The squares show the results of the PBD algorithm. Each point is an average over 50 networks.

3.1. Computer-generated Networks

To conclude, we include experiments carried out with the computer-generated networks first proposed by Girvan and Newman [11], which have become a common testbed in the field. Those networks are constructed with 128 nodes divided into four groups of the same size. For each node 8 edges are deployed. With probability P_{in} the edge is connected to a node that belongs to the same group chosen at random. Otherwise, the edge is connected to a node that does not belong to the same group. Thus, the average degree of a node is 16. Accordingly to the nomenclature of Girvan-Newman, we will use z_{out}, which is the number of *inter-community* edges per node. It is important to notice that for $z_{out} = 12$ the network is totally random, that is, without community structure. Another important value for z_{out} is 8, since it marks the boundary between having more *inter-community* than *intra-community* edges. The quality measurement is the fraction of vertices that are correctly classified, explained in more detail in [14].

In figure 6 we can see the results obtained by our algorithm compared to the results yield by the Girvan-Newman (GN) algorithm based on edge betweenness [11]. We decided to use the GN algorithm instead of the Newman's Fast algorithm since it obtains slightly better results and it is the reference algorithm for this particular experiment [11]. Although the results of GN outperform those obtained by the N algorithm it is not a suitable option for medium and large networks due to its complexity, that is $O(n^3)$.

The GN algorithm correctly detects the communities until values of $z_{out} = 6$ are reached. From this point on, the quality of the communities decreases very quickly. On the other hand, the PBD algorithm detects the communities very well up to values of $z_{out} = 7$, from that point on the performance starts to decay, although the pace is more steady than in the GN case. As already mentioned, other algorithms based on modularity optimization perform much better in the computer-generated networks example. For instance, Duch and Arenas extremal optimization algorithm [16] starts to decline at $z_{out} = 8$. However, its increase on clustering quality is done at the expense of efficiency. So, it is unadvisable in the case of large networks.

4. CONCLUSIONS

In this paper we have presented an algorithm to extract community structure from networks by using a combination of different existing methods. First, the algorithm uses spectral analysis via the multiple random walker process to reduce the dimensionality of the network by creating the initial partition of the network into communities. Then, a modularity optimization process with a extremely greedy search heuristic is applied to extract the underlying structure of the network.

Experiments show that our algorithm outperforms Newman's Fast algorithm both in clustering quality and efficiency. Newman's Fast algorithm is the reference algorithm in terms of efficiency, and while asymptotically both algorithms are $O(n^2)$, PDB algorithm is always faster in computation time for medium and large networks, as it has been shown in the experiments.

The reason behind this is the reduction of dimensionality provided by the random walker process, such that the cost of PBD can be expressed as $O(ns)$ where $s \ll n$. Furthermore, experiments also show that PDB retrieves more *general* community structure than other algorithms. The number of existing communities in the partition with maximum modularity is notably smaller in the case of PBD. This fact leads us to think that other algorithms tend to unnecessarily over-specialize their clustering.

In summary, the presented algorithm is an interesting choice when analyzing medium and large networks. The structure of large networks can be found in reasonable time, from seconds for a network of 27k vertices to less than one hour for a network of 500k vertices. The gain in efficiency does not come with a loss in the quality of the clustering as the maximum modularity obtained by the algorithm is comparable to the reference algorithms in the literature.

Acknowledgments

The authors would like to thank Jordi Duch and Alex Arenas for his helpful comments and for providing us with the binary of their EO algorithm [16]. We also thank Mark Newman, Fabrizio Ferraro and Mariano Belinsky for providing some of the datasets used in the experiment. This is also extended to those researchers that compiled and made public the remaining datasets used in the experiments. This research was supported by the Spanish Government Grant CICYT-TIC2003-1120-C03-02.

[1] R. Albert, H. Jeong, and A.-L. Barabási, Nature pp. 130–131 (1999).
[2] F. Liljeros, C. Edling, L. Amaral, H. Stanley, and Y. Aberg, Nature **411**, 907 (2001).
[3] B. A.-L., H. Jeong, E. Ravasz, Z. Néda, A. Schubert, and T. Vicsek, Tech. Rep. 0104162, ArXiv:cond-mat (2001).
[4] M. Newman, Phys. Rev. E **64**, 016131 (2000).
[5] H. Jeong, S. Mason, Z. Oltvai, and A.-L. Barabási, Nature **411**, 41 (2001).
[6] R. Albert, H. Jeong, and A.-L. Barabási, Nature **406**, 378 (2000).
[7] W. Aiello, F. Chung, and L. Lu, in *Proceedings of the 32nd ACM Symposium on the Theory of Computing*, edited by F. Yao (ACM, 2000), pp. 171–180.
[8] R. Albert and A.-L. Barabási, Review of Modern Physics **74**, 47 (2002).
[9] M. Newman, SIREV **45**, 167 (2003).
[10] D. Watts and S. Strogatz, Nature **393**, 440 (1998).
[11] M. Girvan and M. Newman, Proceedings of the National Academy of Science **99**, 7821 (2002).
[12] R. Kannan, S. Vampala, and A. Vetta, in *Foundations of Computer Science* (2000), pp. 367–378.
[13] M. Newman and M. Girvan, Phys. Rev. E **69**, 026113 (2004).
[14] M. Newman, Phys. Rev. E **69**, 066113 (2004).
[15] R. Guimerà and L. Amaral, Nature **433**, 895 (2005).
[16] J. Duch and A. Arenas, Phys. Rev. E **72**, 027104 (2005).
[17] M. Newman, arxiv:physics/0602124 (2006).
[18] L. Danon, A. Díaz-Guilera, and A. Arenas, arxiv:physics/0601144 (2006).
[19] S. Wasserman and K. Faust, *Social Network Analysis: Methods and Applications* (Cambridge University Press, 1994).
[20] D. Gibson, J. Kleinberg, and P. Raghavan, in *Proceedings of the ACM Symposium on Hypertext and Hypermedia* (1998).
[21] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, in *Proceedings of the 8th World Wide Web Conference* (1999).
[22] C. Borgs, J. Chayes, M. Mahdian, and A. Saberi, in *Proceedings of KDD* (2004), pp. 783–787.
[23] D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, et al., Nucleic Acids Research **91**, 2443 (2003).
[24] H. D. Simon, Computing Systems in Engineering **2**, 135 (1991).
[25] L. Page, S. Brin, R. Motowani, and T. Winograd, Tech. Rep. 1997-0072, Stanford Digital Library Working Paper (1997).
[26] J. Kleinberg, Tech. Rep. RJ 10076, IBM (1997).
[27] G. Flake, R. Tarjan, and K. Tsioutsiouliklis, Internet Mathematics **1**, 385 (2003).
[28] D. Harel and Y. Koren, Lecture Notes in Computer Science **2245**, 18 (2001).
[29] U. Brandes, M. Gaertler, and D. Wagner, Lecture Notes in Computer Science **2832**, 568 (2003).
[30] A. Jain and R. Dubes, *Algorithms for Clustering Data* (Prentice-Hall, 1988).
[31] A. Jain, M. Murty, and P. Flynn, ACM Computing Surverys **31** (1999).
[32] E. Forgy, Biometrics **21**, 768 (1965).
[33] P. Bradley and U. Fayyad, in *Proceedings of the 15th Intl. Conf. on Machine Learning* (1998).
[34] W. W. Zachary, Journal of Anthropological Research **33** (1977).
[35] R. Sangüesa and J. Pujol, *Knowledge Management and Organizational Memories* (Kluwer Academic Publishers, 2002), chap. NetExpert: Agent-based Expertise Location by Means of Social and Knowledge Networks.
[36] H. Jeong, B. Tombor, R. Albert, Z. Oltvai, and A.-L. Barabási, Nature **407**, 651 (2000).
[37] A.-L. Barabási and R. Albert, Science **286**, 509 (1999).
[38] M. Gustafsson, M. Hörnquist, and A. Lombardi, Physica A **In Press** (2006).
[39] D. Gusfield, Inform. Process. Lett. **3**, 159 (2002).
[40] Erdös Number Project. The network contains scientists with Erdös number less than or equal to 2 up to year 2002. http://www.oakland.edu/enp/thedata.html
[41] Network from Garfields's collection of citation networks. Available at http://www.garfield.library.upenn.edu/histcomp/index.html
[42] Data provided by Prof Fabrizio Ferraro, from the project *Small Worlds Of Corporate Networks* at IESE Business School, University of Navarra
[43] Network obtained from the Pajek Network dataset. Available at http://vlado.fmf.uni-lj.si/pub/networks/data/
[44] This network is a bipartite network (two-mode). Conse-

quently, vertices have different roles and thus the structure extracted by the algorithms might be biased. We did not transform it to a one-mode network to maintain the maximum of vertices. Thus, the results on this network are only relevant when speaking about the algorithm's efficiency