Computing Communities in Large Networks Using Random Walks

Pascal Pons and Matthieu Latapy

LIAFA – Université Paris Denis Diderot and CNRS
2 place Jussieu, F-75251 Paris Cedex 05, France
{pons, latapy}@liafa.jussieu.fr

Abstract. Dense subgraphs of sparse graphs (*communities*), which appear in most real-world complex networks, play an important role in many contexts. Computing them however is generally expensive. We propose here a measure of similarities between vertices based on random walks which has several important advantages: it captures well the community structure in a network, it can be computed efficiently, it works at various scales, and it can be used in an agglomerative algorithm to compute efficiently the community structure of a network. We propose such an algorithm which runs in time $O(mn^2)$ and space $O(n^2)$ in the worst case, and in time $O(n^2 \log n)$ and space $O(n^2)$ in most real-world cases (n and m are respectively the number of vertices and edges in the input graph).

1 Introduction

Recent advances have brought out the importance of *complex networks* in many different domains such as sociology (acquaintance or collaboration networks), biology (metabolic networks, gene networks) or computer science (Internet topology, Web graph, P2P networks). We refer to [1,2,3,4,5] for reviews from different perspectives and for an extensive bibliography. The associated graphs are in general globally sparse but locally dense: there exist groups of vertices, called *communities*, highly connected between them but with few links to other vertices. This kind of structure brings out much information about the network.

This notion of community is however difficult to define formally. Many definitions have been proposed in social networks studies [1], but they are too restrictive or cannot be computed efficiently. However, most recent approaches have reached a consensus, and consider that a partition $\mathcal{P} = \{C_1, \ldots, C_k\}$ of the vertices of a graph $G = (V, E)$ ($\forall i, C_i \subseteq V$) represents a good community structure if the proportion of edges inside the C_i (internal edges) is high compared to the proportion of edges between them. Therefore, we will design an algorithm which finds communities satisfying this criterion.

We will consider throughout this paper an *undirected graph* $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges. We impose that each vertex is linked to itself by a loop (we add these loops if necessary). We also suppose that G is connected, the case where it is not being treated by considering the components as different graphs.

P. Yolum et al.(Eds.): ISCIS 2005, LNCS 3733, pp. 284–293, 2005.

1.1 Our Approach and Results

Our approach is based on the following intuition: random walks on a graph tend to get "trapped" into densely connected parts corresponding to communities. We therefore begin with a theoretical study of random walks on graphs. Using this, we define a measurement of the structural similarity between vertices and between communities, thus defining a distance. We relate this distance to existing spectral approaches of the problem. But our distance has an important advantage on these methods: it is efficiently computable, and can be used in a hierarchical clustering algorithm (merging iteratively the vertices into communities). One obtains this way a hierarchical community structure that may be represented as a tree structure called *dendrogram* (an example is provided in Figure 1). We propose such an algorithm which computes a community structure in time $\mathcal{O}(mnH)$ where H is the height of the corresponding dendrogram. The worst case is $\mathcal{O}(mn^2)$. But most real-world complex networks are sparse ($m = \mathcal{O}(n)$) and, as already noticed in [6], H is generally small and tends to the most favourable case in which the dendrogram is balanced ($H = \mathcal{O}(\log n)$). In this case, the complexity is therefore $\mathcal{O}(n^2 \log n)$.

1.2 Related Work

Community detection is related to the classical problem of *graph partitioning* that consists in splitting a graph into a given number of groups while minimizing the cost of the edge cut [7,8]. However, these algorithms are not well suited to our case because they need the number of communities and their size as parameters. The recent interest in the domain has started with a new *divisive* approach proposed by Girvan and Newman [9,10]: the edges with the largest *betweenness* are removed one by one in order to split hierarchically the graph into communities. This algorithm runs in time $\mathcal{O}(m^2n)$. Similar algorithms were proposed by Radicchi *et al* [11] and by Fortunato *et al* [12]. The first one uses a local quantity (the number of loops of a given length containing an edge) to choose the edges to remove and runs in time $\mathcal{O}(m^2)$. The second one uses a more complex notion of information centrality with a time complexity $\mathcal{O}(m^3n)$.

Hierarchical clustering is another classical approach: from a measurement of the similarity between vertices, an *agglomerative* algorithm groups iteratively the vertices into communities (different methods exist, differing on the way of choosing the communities to merge at each step). Several agglomerative methods have been recently introduced. Newman proposed in [13] a greedy algorithm that starts with n communities corresponding to the vertices and merges communities in order to optimize a function called modularity which measures the quality of a partition. This algorithm runs in $\mathcal{O}(mn)$ and has recently been improved to a complexity $\mathcal{O}(mH \log n)$ (with our notations) [6]. The algorithm of Donetti and Muñoz [14] uses the eigenvectors of the Laplacian matrix of the graph to measure the similarities between vertices. The complexity is determined by the computation of all the eigenvectors, in $\mathcal{O}(n^3)$ time for sparse matrices. Other interesting methods have been proposed, see for instance [15,16,17,18].

Random walks have already been used to infer structural properties of networks in some previous works. Gaume [19] used this notion in linguistic context. Fouss *et* al [20] used the Euclidean commute time distance based on the average first-passage time of walkers. Zhou and Lipowsky [21] introduced another dissimilarity index basd on the same quantity, it has been integrated in a hierarchical algorithm (*Netwalk*). *Markov Cluster Algorithm* [22] iterates two matrix operations (one corresponding to random walks) bringing out clusters in the limit state. Unfortunately the three last approaches runs in $\mathcal{O}(n^3)$ and cannot manage networks with more than a few thousand vertices. Our approach has the main advantage to be significatively faster while producing very good results.

2 Preliminaries on Random Walks

The graph G is associated with its *adjacency matrix* A: $A_{ij} = 1$ if vertices i and j are connected and $A_{ij} = 0$ otherwise. The degree $d(i) = \sum_j A_{ij}$ of the vertex i is the number of its neighbors (including itself). To simplify the notations, we only consider unweighted graphs in this paper. It is however trivial to extend our results to weighted graphs ($A_{ij} \in \mathbb{R}^+$ instead of $A_{ij} \in \{0, 1\}$).

Let us consider a discrete *random walk process* (or diffusion process) on the graph G (see [23] for a complete presentation of the topic). At each time step a walker is on a vertex and moves to a vertex chosen randomly and uniformly among its neighbors. The sequence of visited vertices is a *Markov chain*, the states of which are the vertices of the graph. At each step, the transition probability from vertex i to vertex j is $P_{ij} = \frac{A_{ij}}{d(i)}$. This defines the *transition matrix* P of the random walk.

The process is driven by the powers of the matrix P: the probability of going from i to j through a random walk of length t is $(P^t)_{ij}$. In the following, we will denote this probability by P^t_{ij}. It satisfies two general properties of the random walk process which we will use in the sequel:

Property 1. When the length t of a random walk starting at vertex i tends towards infinity, the probability of being on a vertex j only depends on the degree of vertex j (and not on the starting vertex i): $\forall i, \lim_{t\to+\infty} P^t_{ij} = \frac{d(j)}{\sum_k d(k)}$.

Property 2. The probabilities of going from i to j and from j to i through a random walk of a fixed length t have a ratio that only depends on the degrees $d(i)$ and $d(j)$: $\forall i, \forall j, d(i)P^t_{ij} = d(j)P^t_{ji}$.

3 Comparing Vertices Using Short Random Walks

In order to group the vertices into communities, we will now introduce a distance r between the vertices that captures the community structure of the graph. This distance must be large if the two vertices are in different communities, and on the contrary if they are in the same community it must be small. It will be computed from the information given by random walks in the graph.

Let us consider random walks on G of a given length t. We will use the information given by all the probabilities P_{ij}^t to go from i to j in t steps. The length t of the random walks must be sufficiently long to gather enough information about the topology of the graph. However t must not be too long, to avoid the effect predicted by Property 1; the probabilities would only depend on the degree of the vertices. Each probability P_{ij}^t gives some information about the two vertices i and j, but Property 2 says that P_{ij}^t and P_{ji}^t encode exactly the same information. Finally, the information about vertex i encoded in P^t resides in the n probabilities $(P_{ik}^t)_{1\leq k\leq n}$, which is nothing but the i^{th} row of the matrix P^t, denoted by $P_{i\bullet}^t$. To compare two vertices i and j using these data, we must notice that:

- If two vertices i and j are in the same community, the probability P_{ij}^t will surely be high. But the fact that P_{ij}^t is high does not necessarily imply that i and j are in the same community.
- The probability P_{ij}^t is influenced by the degree $d(j)$ because the walker has higher probability to go to high degree vertices.
- Two vertices of a same community tend to "see" all the other vertices in the same way. Thus if i and j are in the same community, we will probably have $\forall k, P_{ik}^t \simeq P_{jk}^t$.

We can now give the definition of our distance between vertices, which takes into account all previous remarks:

Definition 1. *Let i and j be two vertices in the graph and*

$$r_{ij} = \sqrt{\sum_{k=1}^{n} \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}} = \left\| D^{-\frac{1}{2}} P_{i\bullet}^t - D^{-\frac{1}{2}} P_{j\bullet}^t \right\| \tag{1}$$

where $\|.\|$ is the Euclidean norm of $\mathbb{R}^n$.

One can notice that this distance can also be seen as the L^2 distance between the two probability distributions $P_{i\bullet}^t$ and $P_{j\bullet}^t$. Notice also that the distance depends on t and may be denoted $r_{ij}(t)$. We will however consider it as implicit to simplify the notations.

Theorem 1. *The distance r is related to the spectral properties of the matrix P*

$$r_{ij}^2 = \sum_{\alpha=2}^{n} \lambda_\alpha^{2t} (v_\alpha(i) - v_\alpha(j))^2$$

where $(\lambda_\alpha)_{1\leq\alpha\leq n}$ and $(v_\alpha)_{1\leq\alpha\leq n}$ are respectively the eigenvalues and right eigenvectors of the matrix P.

This theorem relates random walks on graphs to the many current works that study spectral properties of graphs. For example, [24] notices that the modular structure of a graph is expressed in the eigenvectors of P (other than v_1) that corresponds to the largest positive eigenvalues. If two vertices i and j belong

to a same community then the coordinates $v_\alpha(i)$ and $v_\alpha(j)$ are similar in all these eigenvectors. Moreover, [25,26] show in a more general case that when an eigenvalue λ_α tends to 1, the coordinates of the associated eigenvector v_α are constant in the subsets of vertices that correspond to communities. A distance similar to ours (but that cannot be computed directly with random walks) is also introduced: $d_t^2(i,j) = \sum_{\alpha=2}^{n} \frac{(v_\alpha(i)-v_\alpha(j))^2}{1-|\lambda_\alpha|^t}$. Finally, [14] uses the same spectral approach applied to the Laplacian matrix of the graph $L = D - A$.

All these studies show that the spectral approach takes an important part in the search for community structure in graphs. However all these approaches have the same drawback: the eigenvectors need to be explicitly computed (in time $\mathcal{O}(n^3)$ for a sparse matrix). This computation rapidly becomes untractable in practice when the size of the graph exceeds some thousands of vertices. Our approach is based on the same foundation but has the advantage of avoiding the expensive computation of the eigenvectors: it only needs to compute the probabilities P_{ij}^t, which can be done efficiently as shown in the following theorem.

Theorem 2. *All the probabilities P_{ij}^t can be computed in time $\mathcal{O}(tnm)$ and space $\mathcal{O}(n^2)$. Once these probabilities computed, each distance r_{ij} can be computed in time $\mathcal{O}(n)$.*

Proof. To compute the vector $P_{i\bullet}^t$, we multiply t times the vector $P_{i\bullet}^0$ ($\forall k$, $P_{i\bullet}^0(k) = \delta_{ik}$) by the matrix P. This direct method is advantageous because the matrix P is generally sparse (for real-world complex networks) therefore each product is processed in time $\mathcal{O}(m)$. The initialization of $P_{i\bullet}^0$ is done in $\mathcal{O}(n)$ and thus each of the n vectors $P_{i\bullet}^t$ is computed in time $\mathcal{O}(n+tm) = \mathcal{O}(tm)$. Once we have the two vectors $P_{i\bullet}^t$ and $P_{j\bullet}^t$, we can compute r_{ij} in $\mathcal{O}(n)$ using Equation (1).

Now we generalize our distance between vertices to a distance between communities in a straightforward way. Let us consider random walks that start from a community: the starting vertex is chosen randomly and uniformly among the vertices of the community. We define the probability P_{Cj}^t to go from community C to vertex j in t steps:

$$P_{Cj}^t = \frac{1}{|C|} \sum_{i \in C} P_{ij}^t$$

This defines a probability vector $P_{C\bullet}^t$ that allows us to generalize our distance:

Definition 2. *Let $C_1, C_2 \subset V$ be two communities. We define the distance $r_{C_1C_2}$ between these two communities by:*

$$r_{C_1C_2} = \left\| D^{-\frac{1}{2}} P_{C_1\bullet}^t - D^{-\frac{1}{2}} P_{C_2\bullet}^t \right\| = \sqrt{\sum_{k=1}^{n} \frac{(P_{C_1k}^t - P_{C_2k}^t)^2}{d(k)}}$$

This definition is consistent with the previous one: $r_{ij} = r_{\{i\}\{j\}}$ and we can also define the distance between a vertex and a community. Given the probability vectors $P_{C_1\bullet}^t$ and $P_{C_2\bullet}^t$, the distance $r_{C_1C_2}$ is also computed in time $\mathcal{O}(n)$.

4 The Algorithm

In the previous section, we have proposed a distance between vertices (and between sets of vertices) to capture structural similarities between them. The problem of finding communities is now a clustering problem. We will use here an efficient hierarchical clustering algorithm that allows us to find community structures at different scales. We present an agglomerative approach based on Ward's method [27] that is well adapted to our distance and gives very good results while reducing the number of distance computations in order to be able to process large graphs.

We start from a partition $\mathcal{P}_1 = \{\{v\}, v \in V\}$ of the graph into n communities reduced to a single vertex. We first compute the distances between all adjacent vertices. Then this partition evolves by repeating the following operations. At each step k:

- Choose two communities C_1 and C_2 in $\mathcal{P}_k$ on a criterion based on the distance between the communities that we detail later.
- Merge these two communities into a new community $C_3 = C_1 \cup C_2$ and create the new partition: $\mathcal{P}_{k+1} = (\mathcal{P}_k \setminus \{C_1, C_2\}) \cup \{C_3\}$.
- Update the distances between communities (we will see later that we actually only do this for *adjacent* communities).

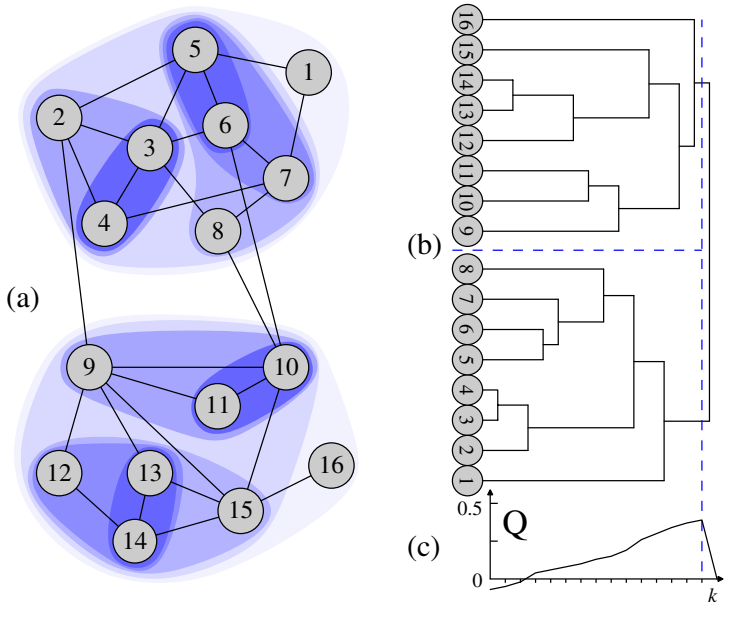

Fig. 1. (a) An example of community structure found by our algorithm using random walks of length $t = 3$. (b) The stages of the algorithm encoded as a tree (*dendrogram*). The maximum of Q, plotted in (c), shows that the best partition consists in two communities.

After $n-1$ steps, the algorithm finishes and we obtain $\mathcal{P}_n = \{V\}$. Each step defines a partition $\mathcal{P}_k$ of the graph into communities, which gives a hierarchical structure of communities called dendrogram (see Figure 1(b)). This structure is a tree in which the leaves correspond to the vertices and each internal node is associated with a merging of communities in the algorithm: it corresponds to a community composed of the union of the communities corresponding to its children. The key points in this algorithm are the way we choose the communities to merge, and the fact that the distances can be updated efficiently. We will also need to evaluate the quality of a partition in order to choose one of the $\mathcal{P}_k$ as the result of our algorithm. We will detail these points below, and explain how they can be managed to give an efficient algorithm.

Choosing the communities to merge. This choice plays a central role for the quality of the community structure created. In order to reduce the complexity, we will only merge *adjacent* communities (having at least an edge between them). This reasonable heuristic (already used in [13] and [14]) limits to m the number of possible mergings at each stage. Moreover it ensures that each community is connected. We choose the two communities to merge according to Ward's method. At each step k, we merge the two communities that minimize the mean σ_k of the squared distances between each vertex and its community.

$$\sigma_k = \frac{1}{n} \sum_{C \in \mathcal{P}_k} \sum_{i \in C} r_{iC}^2$$

This approach is a greedy algorithm that tries to solve the problem of maximizing σ_k for each k. But this problem is known to be NP-hard: even for a given k, maximizing σ_k is the NP-hard "K-Median clustering problem". So for each pair of adjacent communities $\{C_1, C_2\}$, we compute the variation $\Delta\sigma(C_1, C_2)$ of σ if we would merge C_1 and C_2 into a new community $C_3 = C_1 \cup C_2$ and we merge the two communities that give the lowest value of $\Delta\sigma$. This quantity only depends on the vertices of C_1 and C_2, and not on the other communities or on the step k of the algorithm:

$$\Delta\sigma(C_1, C_2) = \frac{1}{n}\Big(\sum_{i \in C_3} r_{iC_3}^2 - \sum_{i \in C_1} r_{iC_1}^2 - \sum_{i \in C_2} r_{iC_2}^2 \Big) \tag{2}$$

Computing $\Delta\sigma$ and updating the distances. The important point here is to notice that these quantities can be efficiently computed thanks to the fact that our distance is a Euclidean distance, which makes it possible to obtain the following classical result [28]:

Theorem 3. *The increase of σ after the merging of two communities C_1 and C_2 is directly related to the distance $r_{C_1C_2}$ by:*

$$\Delta\sigma(C_1, C_2) = \frac{1}{n} \frac{|C_1||C_2|}{|C_1| + |C_2|} r_{C_1C_2}^2$$

This theorem shows that we only need to update the distances between communities to get the values of $\Delta\sigma$: if we know the two vectors $P_{C_1\bullet}$ and $P_{C_2\bullet}$, the computation of $\Delta\sigma(C_1, C_2)$ is possible in $\mathcal{O}(n)$. Since we only merge adjacent communities, we only need to update the values of $\Delta\sigma$ between adjacent communities (there are at most m values). These values are stored in a balanced tree in which we can add, remove or get the minimum in $\mathcal{O}(\log m)$.

Evaluating the quality of a partition. The algorithm induces a sequence $(\mathcal{P}_k)_{1\leq k\leq n}$ of partitions into communities. We now want to know which partitions in this sequence capture well the community structure. The most widely used method is to choose the partition maximizing the modularity Q introduced in [10,13]. This quantity uses the fraction of edges e_C that are inside the community C and the fraction of edges[1] a_C bound to the community C: $Q(\mathcal{P}) = \sum_{C\in\mathcal{P}} e_C - a_C^2$. However, depending on one's objectives, one may consider other quality criterion.

Complexity. First, the initialization of the probability vectors is done in $\mathcal{O}(mnt)$. Then, at each step k of the algorithm, we keep in memory the vectors $P^t_{C\bullet}$ corresponding to the current communities (the ones in the current partition). But for the communities that are not in $\mathcal{P}_k$ (because they have been merged with another community before) we only keep the information saying in which community it has been merged. We keep enough information to construct the dendogram and have access to the composition of any community with a few more computation.

When we merge two communities C_1 and C_2 we perform the following operations:

- Compute $P^t_{(C_1\cup C_2)\bullet} = \frac{|C_1|P^t_{C_1\bullet} + |C_2|P^t_{C_2\bullet}}{|C_1|+|C_2|}$ and remove $P^t_{C_1\bullet}$ and $P^t_{C_2\bullet}$.
- Update the values of $\Delta\sigma$ concerning C_1 and C_2 using Theorem 3.

The first operation can be done in $\mathcal{O}(n)$, and therefore does not play a significant role in the overall complexity of the algorithm. The dominating factor in the complexity of the algorithm is the number of distances r computed (each one in $\mathcal{O}(n)$). We prove an upper bound of this number that depends on the height H of the dendrogram.

Theorem 4. *An upper bound of the number of distances computed by our algorithm is $2mH$. Therefore its global time complexity is $\mathcal{O}(mn(H + t))$.*

In practice, a small t must be chosen (we must have $t = \mathcal{O}(\log n)$ due to the exponential convergence speed of the random walks process) and thus the global complexity is $\mathcal{O}(mnH)$. The worst case is $H = n - 1$, which occurs when the vertices are merged one by one to a large community. This happens in the "star" graph, where a central vertex is linked to the $n - 1$ others. However Ward's algorithm is known to produce small communities of similar sizes. This tends to get closer to the favorable case in which the community structure is a balanced tree and its height is $H = \mathcal{O}(\log n)$.

[1] Inter-community edges contribute for $\frac{1}{2}$ to each community.

5 Conclusion and Further Work

We proposed a new distance between the vertices that quantify their structural similarities using random walks. This distance has several advantages: it captures much information on the community structure, it is well suited for approximation, and it can be used in an efficient hierarchical agglomerative algorithm. We designed such an algorithm which works in time $\mathcal{O}(mnH)$. In practice, real-world complex networks are sparse ($m = \mathcal{O}(n)$) and the height of the dendrogram is generally small ($H = \mathcal{O}(\log n)$); in this case the algorithm runs in $\mathcal{O}(n^2 \log n)$. An implementation is provided at [29].

We presented in this short paper the main principle of the algorithm. However many improvements and optimizations have been implemented, which make it possible to process very large networks (up to several hundred thousand vertices). Moreover, extensive experiments have been run to compare the different existing approaches. They show that our method provides excellent results in different conditions (graph sizes, densities, number of communities, and community size distributions) while having a time complexity among the best ones.

We are convinced that our method could be integrated in a multi-scale visualization tool for large networks. Our approach may also be relevant for the computation of *overlapping* communities (which often occurs in real-world cases and is not considered by any algorithm until now). We consider these two points as promising directions for further work. Finally, we pointed out that the method is directly usable for *weighted* networks. For directed ones (like the important case of the Web graph), on the contrary, the proofs we provided are not valid anymore, and random walks behave significantly differently. Therefore, we also consider the directed case as an interesting direction.

Acknowledgments. We thank Annick Lesne and L.S. Shulman for useful conversation and Clémence Magnien for helpful comments on preliminary versions. This work has been supported in part by the PERSI (*Programme d'Étude des Réseaux Sociaux de l'Internet*) project and by the GAP (*Graphs, Algorithms and Probabilities*) project.

References

1. Wasserman, S., Faust, K.: Social network analysis. Cambridge University Press, Cambridge (1994)
2. Strogatz, S.H.: Exploring complex networks. Nature **410** (2001) 268–276
3. Albert, R., Barabási, A.L.: Statistical mechanics of complex networks. Reviews of Modern Physics **74** (2002) 47
4. Newman, M.E.J.: The structure and function of complex networks. SIAM REVIEW **45** (2003) 167
5. Dorogovtsev, S., Mendes, J.: Evolution of Networks: From Biological Nets to the Internet and WWW. Oxford University Press, Oxford (2003)
6. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Physical Review E **70** (2004) 066111

7. Pothen, A., Simon, H.D., Liou, K.P.: Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Matrix Anal. Appl. **11** (1990) 430–452
8. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell System Technical Journal **49** (1970) 291–308
9. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. PNAS **99** (2002) 7821–7826
10. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69** (2004) 026113
11. Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., Parisi, D.: Defining and identifying communities in networks. PNAS **101** (2004) 2658–2663
12. Fortunato, S., Latora, V., Marchiori, M.: Method to find community structures based on information centrality. Physical Review E **70** (2004) 056104
13. Newman, M.E.J.: Fast algorithm for detecting community structure in networks. Physical Review E **69** (2004) 066133
14. Donetti, L., Muñoz, M.A.: Detecting network communities: a new systematic and efficient algorithm. Journal of Statistical Mechanics **2004** (2004) 10012
15. Wu, F., Huberman, B.A.: Finding communities in linear time: A physics approach. The European Physical Journal B **38** (2004) 331–338
16. Reichardt, J., Bornholdt, S.: Detecting fuzzy community structures in complex networks with a potts model. Physical Review Letters **93** (2004) 218701
17. Bagrow, J., Bollt, E.: A local method for detecting communities. Physical Review E (2005 (to appear))
18. Duch, J., Arenas, A.: Community detection in complex networks using extremal optimization. arXiv:cond-mat/0501368 (2005)
19. Gaume, B.: Balades alatoires dans les petits mondes lexicaux. I3 Information Interaction Intelligence **4** (2004)
20. F., F., A., P., M., S.: A novel way of computing dissimilarities between nodes of a graph, with application to collaborative filtering. In: Workshop on Statistical Approaches for Web Mining (SAWM), Pisa (2004) 26–37
21. Zhou, H., Lipowsky, R.: Network brownian motion: A new method to measure vertex-vertex proximity and to identify communities and subcommunities. In: International Conference on Computational Science. (2004) 1062–1069
22. van Dongen, S.: Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht (2000)
23. Lovász, L.: Random walks on graphs: a survey. In: Combinatorics, Paul Erdős is eighty, Vol. 2 (Keszthely, 1993). Volume 2 of Bolyai Soc. Math. Stud. János Bolyai Math. Soc., Budapest (1996) 353–397
24. Simonsen, I., Eriksen, K.A., Maslov, S., Sneppen, K.: Diffusion on complex networks: a way to probe their large-scale topological structures. Physica A: Statistical Mechanics and its Applications **336** (2004) 163–173
25. Schulman, L.S., Gaveau, B.: Coarse grains: The emergence of space and order. Foundations of Physics **31** (2001) 713–731
26. Gaveau, B., Lesne, A., Schulman, L.S.: Spectral signatures of hierarchical relaxation. Physics Letters A **258** (1999) 222–228
27. Ward, J.H.: Hierarchical grouping to optimize an objective function. Journal of the American Statistical Association **58** (1963) 236–244
28. Jambu, M., M.-O., L.: Cluster analysis and data analysis. North Holland Publishing (1983)
29. Pons, P.: (`http://liafa.jussieu.fr/~pons/`)