RAIRO-Oper. Res. 42 (2008) 469–484
DOI: 10.1051/ro:2008029

RAIRO Operations Research
www.rairo-ro.org

COMPARISON OF ALGORITHMS IN GRAPH PARTITIONING *

Alain Guénoche[1]

Abstract. We first describe four recent methods to cluster vertices of an undirected non weighted connected graph. They are all based on very different principles. The fifth is a combination of classical ideas in optimization applied to graph partitioning. We compare these methods according to their ability to recover classes initially introduced in random graphs with more edges within the classes than between them.

Keywords. Graph partitioning, partition comparison, simulation.

Mathematics Subject Classification. 05C85, 90C35, 90C59.

Introduction

The partitioning problem in graphs has a long history we cannot detail here. It becomes important in practice with the pagination of electronic circuits and VLSI design [1]. There are also many contributions linked to the graph drawing problem [4]. The initial aim of these clustering processes was to minimize the number of inter-class edges to disconnect the graph. It has been reactivated these last years in three domains:

- biological problems modelled by graphs as protein interaction networks [2,3,7,13];
- the study of large networks, like WEB [18]; or

Received December 31, 2006. Accepted January 09, 2008.

* *This work has been realized from 2003 to 2006, with the financial help of CNRS (ACI-IMPBio) and many fruitful discussions with several colleagues. A preliminary version has been presented at the International Conference on Graph Theory, Hyères, September 2005.*

[1] IML-CNRS, 163 Av. de Luminy, 13288 Marseille Cedex 9, France;
guenoche@lim.univ-mrs.fr

Article published by EDP Sciences

- the definition of *communities* in social networks [14,20].

In these domains, the common aim is to put together vertices sharing a large number of edges, making some *high density zones*, compared to the rate of edges observed in the whole graph. Several ideas appeared these last years and lead to algorithms based on different principles:

- the first one is linked to a density function associated to each vertex, which is as large as there are many edges in its neighborhood [9];
- the second one is based on a dynamical weight function associated to the edges, called *betweenness* in the article, and a pruning of the graph until disconnection [14];
- the third one is the result of random walks from any vertex in the graph, which is denoted *Markov clustering (MCL)* by the author van Dongen [11];
- the fourth one is based on a spectral decomposition of a matrix derived from the adjacency matrix [21];
- finally, the fifth one is a classical strategy in clustering, I have adjusted, optimizing a criterion adapted to the graph partitioning problem.

In this text we compare these five methods, applied to undirected none weighted graphs, testing their ability to recover high density zones when they exist. Similar studies appear recently as [20] or [6]. In this article, we realize simulations generating random graphs where such dense classes have been introduced. After partitioning, we evaluate in many ways the gap between the initial partition and the computed one; four criteria are used. According to their average values, one can decide which is the most appropriate algorithm, for graph partitioning.

This is not the usual strategy: often, the authors select a criterion, as the *modularity* (Newman, 2004) which now receives a great attention, and show that their algorithm gives the best results (for this criterion) on very few "real" graphs considered as benchmark. One can think it is not sufficiently convincing, because this appreciation depends on the type of graph and some parameter values, making easy or difficult partitioning problems, and also on the number of tested graphs. That's why we define a precise generating model for graphs, with an initial clear partition to recover and several criteria corresponding to many aspects of partition's similarity, combining two types of algorithms, those for which the number of classes is fixed and those for which it is free.

Notations are as follows: let X be a set of n vertices, E the set of m edges and $\Gamma = (X, E)$ the corresponding graph. It is assumed to be connected; otherwise its different components are handled separately. For any part Y of X, let $\Gamma(Y)$ be the set of vertices out of Y that are adjacent to Y

$$\Gamma(Y) = \{x \in X \setminus Y \text{ such that } \exists y \in Y, (x, y) \in E\},$$

and $\overline{\Gamma(Y)} = Y \bigcup \Gamma(Y)$. The neighborhood of x is $\Gamma(x)$. The degree of vertex x is denoted $Dg(x) = |\Gamma(x)|$ and let δ be the maximum degree in the graph. The internal edge set of a class $Y \subset X$ is denoted

$$E(Y) = \{(x, y) \in E \text{ such that } x \in Y \text{ and } y \in Y\}.$$

1. Clustering by density

Clustering methods based on density have been introduced by Wishart in 1976; the initial idea was to build classes around elements having many neighbors in a threshold graph associated to a distance on X. It has not been largely developed because, the simple degree was the single proposed density function, which gives poor results, even if so, non convex or nested clusters can be recovered. Recently, several authors – Rougemont & Hingamp [24] for simple graphs and Guénoche [16] for distance arrays – have reactivated this approach.

1.1. Density function

A density function De is a map from X to $\mathbb{R}_+$ varying increasingly with the number of vertices close to an element. In [9] several functions based on the percentage of edges in the neighborhood of a vertex have been compared. The *core index* introduced by Seidman [25] is also a typical density function. Let us recall that a vertex x has a core value equal to the maximum k for which there exist at least k vertices in $\Gamma(x)$ having also a core index larger than or equal to k. A maximal clique of p elements has core $p-1$ and all the vertices in trees have core 1. The core index can be calculated pruning recursively the graph from minimum degree vertices, and a $O(m)$ algorithm has been proposed by Batagelj and Zaveršnik [5].

All the simulations realized with this kind of density functions are disappointing, mainly because the density values are not spread enough, especially for the core index. For this latter, the connected sets of nodes having the maximum core value are not included in the initial classes of the random graphs (*cf.* Sect. 6). To recover the more satisfying results obtained for distance matrices [16], we first evaluate the Czekanovski-Dice distance between vertices. Its formula is

$$D(x,y) = \frac{|\Delta(\overline{\Gamma}(x), \overline{\Gamma}(y))|}{|\overline{\Gamma}(x)| + |\overline{\Gamma}(y)|}$$

where Δ denotes the symmetrical difference between two sets. We retain this local dissimilarity (which is not a distance) because it is very accurate for graphs since two vertices having no common adjacent vertex are at maximum distance value (equal to 1), and consequently it can be computed in $O(n\delta^3)$, which is linear in the number of vertices, in sparse graph with bounded degre.

The density function in x is then defined from the average distance values between x and its adjacent vertices

$$De(x) = 1 - \frac{\sum_{y\in\Gamma(x)} D(x,y)}{Dg(x)}.$$

TABLE 1. Density values of the Test graph in Figure 1; the local maximum values are printed in bold.

X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
De	**.7**	**.7**	.69	.69	.51	.72	.62	**.8**	.58	.59	**.76**	.6	.72	.49	.71

1.2. Clustering algorithm

The algorithm is in three steps:

First, the *local maximum values* of the density function are considered to identify the *seeds* of the classes. A seed can be a singleton or several connected vertices sharing the same density value. The number of classes, denoted by p, is the number of seeds having a density larger than or equal to the average $\overline{De}$ (another threshold can be chosen to get a number of classes close to the expected one).

In the second step, the seeds are extended: we recursively assign to each one all the connected vertices having a density larger than the average and that are adjacent to only one seed. Doing so, we avoid any ambiguity in the assignment, postponing the decision when several are possible.

In the third step, these seeds are extended to make a partition $(Q_1, ...Q_q)$. Each remaining element x is assigned to one class maximizing criterion C_i:

$$C_i(x) = \frac{|\Gamma(x) \bigcap Q_i|}{Dg(x)}.$$

Compared to the other methods, this algorithm is very fast. To find the local maximum density values is in $O(n\delta^2)$ and the assignment procedure is in $O(n\delta)$. It allows the treatment of large graphs with a low density, ($n \approx 10\,000$). More, the distance matrix D can be computed row after row evaluating the density function, and it is not necessary to store D.

Example 1. All the algorithms will be applied to the graph of Figure 1. Here, the Czekanovski-Dice distance values of the edges are indicated and the density is printed in Table 1, giving an average $\overline{De} = .66$. The local maximum values of the density function are vertices {1, 2}, {8} and {11} making {1, 2, 3, 4}, {6, 8} and {11, 13} as extended seeds. In the third step, elements 7, 8, 9 are assigned to {6, 8} and 10, 12, 14 and 15 go with seed {11, 13}. Finally the partition $(1, 2, 3, 4|5, 6, 7, 8, 9|10, 11, 12, 13, 14, 15)$ is established as it was expected.

2. Disconnecting the graph

The principle of this method has been given by Newman [19]. It consists in an iterative procedure in two steps:

- evaluate the weight $B_e(x, y)$ of any edge (x, y);
- eliminate the edge having the greatest weight.

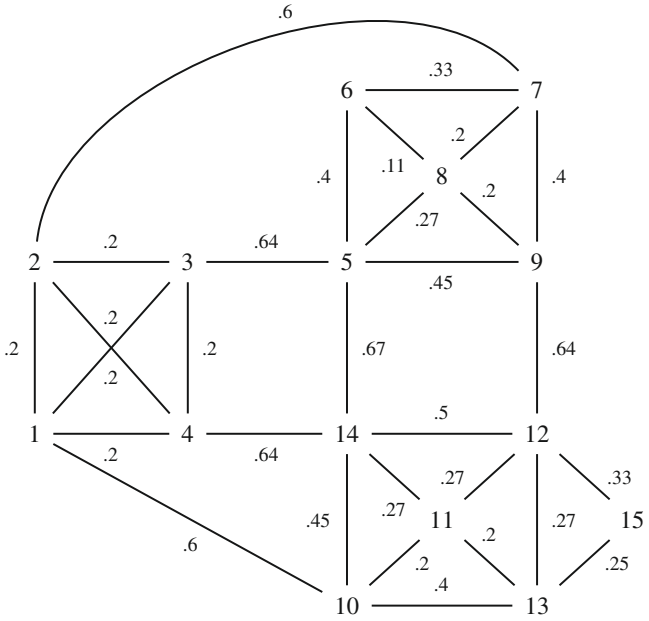

FIGURE 1. Test graph ($n = 15$ and $m = 30$) with the Czekanovski-Dice distance values.

An efficient weight function has been defined by Newman [19], the *edge-betweeness* $B_e : E \mapsto \mathbb{N}$. It corresponds to the number of pairs for which a shortest path, go through the edge (x, y). We have tested:

$$B_e(x, y) = |(z, t) \in X^2 \text{such that} L(x, y) = L(x, z) + 1 + L(t, y)|,$$

where $L(x, y)$ denotes the length of a shortest path between x and y. The weight of an edge is at least equal to 1, which can also be its largest value, as for cliques. Other functions, taking into account the number of shortest paths between any two vertices, as the original betweeness function, have also been tested. It is easy to understand that when there are high density zones, the few paths connecting them receive the largest weights. It suffices to delete the corresponding edges to get dense classes.

The Girwan-Newman method (2002) is a pruning algorithm removing iteratively the edges with maximum weight to shape clusters as connected components. Clearly, these clusters are nested and a complete divisive hierarchy can be established. To use it as a partitioning algorithm, the number of classes must be given. If only p classes are searched, the procedure stops when there are at least p components (there can be more because of ties in the last subdivision). The practical problem of this careful procedure is the time complexity because it could be necessary to perform $O(n^2)$ deletions to get the first subdivision. The author

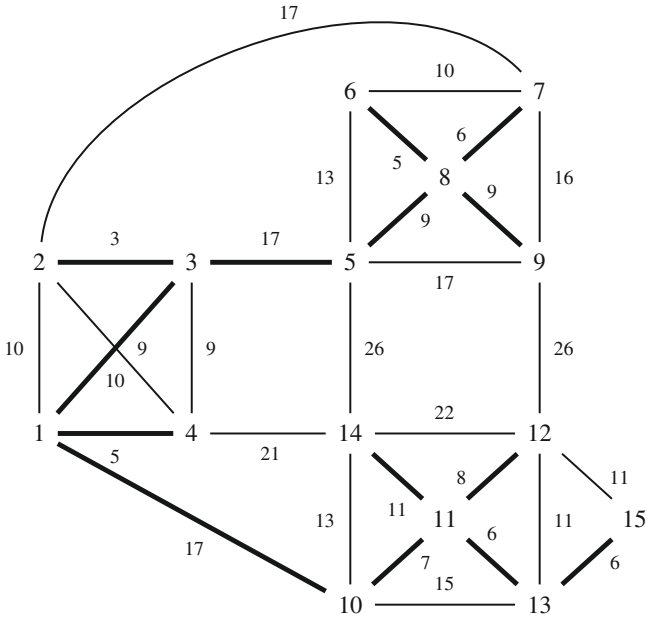

FIGURE 2. Test graph with the betweenness values. The edges of a minimum spanning tree are in bold.

indicates a $O(mn)$ algorithm to evaluate the weights of all the edges, but there are $O(m)$ steps for a complete hierarchy!

Observing that, when there are few connections between what will become separate clusters, removing several edges in one step does not modify the result, we adopt the following strategy:

- evaluate the betweenness function for the edges; let B_{max} be its maximum value;
- build a minimum spanning tree of this weighted graph; let L_{max} be the length of its longest edge;
- remove all the edges longer than or equal to threshold $(L_{max} + B_{max})/2$.

With the L_{max} threshold, the graph would be disconnected in one step. But this rough procedure could give some uncorrect subdivisions. With value $(L_{max} + B_{max})/2$, the split appears when $B_{max} = L_{max}$ and the number of steps is largely reduced. But the average time to establish a partition remains very much longer than the other methods described here.

Example 2. The betweenness values are depicted in Figure 2 and the edges making a minimum spanning tree are enlarged. The maximum is reached for (5, 14) and (9, 12) ($B_{max} = 26$) and the longest edges in the tree are (1, 10) and (3, 5) ($L_{max} = 17$). Removing all the edges longer than or equal to L_{max} would give directly the three expected classes. Here, the careful and the rough strategies lead to the same result.

3. Markov clustering (MCL)

The idea of this method has been given by van Dongen [11]. It is based on the following principle: a random walk in a graph is a path for which the next vertex after x is selected at random in $\overline{\Gamma(x)}$. A random walk from a vertex belonging to a high density zone, remains in the same zone, and the probability to reach another one, after a great number of steps, is very small. Because the next adjacent vertex is selected at random, according to a probability distribution, and as the probability sum to reach them is equal to 1, this walk can be described as a Markov process.

3.1. From the adjacency matrix to a stationary process

Let A be the adjacency matrix of graph Γ ($A(x,y) = 1$ iff $(x,y) \in E$). It is well known that multiplying the adjacency matrix of a graph by itself determines the number of paths of length 2, and so on when the power increases. To ovoid parity dependence on the path length, all the loops are added. Let I denotes the identity matrix; $(A + I)$ is used instead of A.

The Markov matrix M associated to Γ is defined by $M(x,y) = \frac{(A+I)(x,y)}{Dg(y)+1}$. It is column stochastic ($\sum_{x\in X} M(x,y) = 1$), and it can be interpreted for each node (a column y) as a probability to be attracted by its neighbors (row x such that $M(x,y) > 0$). The random walks are simulated at each step by two operators: the *expansion* operator is just the product of M by itself. The *inflation* operator S_r is performed column after column. According to van Dongen, it has been introduced to maintain a stochastic matrix and to reinforce the attracting strength of a row (preserving the values ordering). Acting on column y,

$$M(x,y) \leftarrow \frac{M(x,y)^r}{\sum_{x\in X} M(x,y)^r}.$$

One iteration of the MCL algorithm is to multiply matrix M by itself and to apply the S_r normalization. As any Markov matrix in a finite space, which is irreducible and not periodic, the iterates of M have a limit, which is quickly reached, and the algorithm stops when two consecutive matrices are identical.

3.2. Attractors and classes

The result is a stochastic idempotent matrix M. Often, in a column y there is one element x for which $M(x,y) = 1.0$ and all the other values are equal to 0.0. Element x is said to be the *attractor* of y and, in that case, x is also its own attractor. Very often, there is a value in column y which is close to 1.0 and the complementary part denotes the attraction of another element. Rarely, the attraction is equally shared by several elements to constitute a class of attractors.

In this method, an attractor (or a set of equilibrate attractors) plus the set of attracted elements constitute a class. The number of classes is unpredictable and

TABLE 2. Initial Markov matrix corresponding to the test graph.

	1	10	11	12	13	14	15	2	3	4	5	6	7	8	9
1:	0.20	0.20	0.00	0.00	0.00	0.00	0.00	0.20	0.20	0.20	0.00	0.00	0.00	0.00	0.00
10:	0.20	0.20	0.20	0.00	0.20	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11:	0.00	0.20	0.20	0.17	0.20	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12:	0.00	0.00	0.20	0.17	0.20	0.17	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20
13:	0.00	0.20	0.20	0.17	0.20	0.00	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14:	0.00	0.20	0.20	0.17	0.00	0.17	0.00	0.00	0.00	0.20	0.17	0.00	0.00	0.00	0.00
15:	0.00	0.00	0.00	0.17	0.20	0.00	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2:	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.20	0.20	0.00	0.00	0.20	0.00	0.00
3:	0.20	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.20	0.20	0.17	0.00	0.00	0.00	0.00
4:	0.20	0.00	0.00	0.00	0.00	0.17	0.00	0.20	0.20	0.20	0.00	0.00	0.00	0.00	0.00
5:	0.00	0.00	0.00	0.00	0.00	0.17	0.00	0.00	0.20	0.00	0.17	0.25	0.00	0.20	0.20
6:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.17	0.25	0.20	0.20	0.00
7:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.20	0.00	0.00	0.00	0.25	0.20	0.20	0.20
8:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.17	0.25	0.20	0.20	0.20
9:	0.00	0.00	0.00	0.17	0.00	0.00	0.00	0.00	0.00	0.00	0.17	0.00	0.20	0.20	0.20

TABLE 3. Final idempotent matrix of the Markov process.

	1	10	11	12	13	14	15	2	3	4	5	6	7	8	9
1:	0.47	0.00	0.00	0.00	0.00	0.00	0.00	0.47	0.47	0.47	0.00	0.00	0.00	0.00	0.00
10:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
11:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
12:	0.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
13:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
14:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
15:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2:	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.33	0.33	0.33	0.00	0.00	0.00	0.00	0.00
3:	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.09	0.09	0.09	0.00	0.00	0.00	0.00	0.00
4:	0.11	0.00	0.00	0.00	0.00	0.00	0.00	0.11	0.11	0.11	0.00	0.00	0.00	0.00	0.00
5:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	1.00	1.00	1.00
9:	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

cannot be specified. MCL seems to produce a large number of classes when the rate of edges in the graph is low, but just one class when it is high. Parameter $r > 1$ is not fixed, and it has a great influence on the number of classes (*cf.* Sect. 6). According to Brohée and van Helden [6], a preliminary study to fix r is necessary. Here, we set $r = 2$ which is the best integer value for the Test graph.

Example 3. The initial Markov matrix corresponding to the Test graph is printed in Table 2. Vertex 1 is equally attracted by vertices 1, 10, 2, 3, and 4, ... etc.

After seven iterations we obtained the idempotent matrix of Table 3. The positive values define three connected components that are the final classes: {1, 2, 3, 4}, {10, 11, 12, 13, 14, 15} (all the elements are attracted by 12), and {5, 6, 7, 8, 9} (all attracted by 8).

4. Spectral subdivision

A very new method has been proposed by Newman [21]. The idea is not to maximize the number of intra class edges, but the difference between this number and its expected value "in equivalent networks having edges placed at random". For that, the matrix B derived from the adjacency matrix A is used. The $B(x, y)$ term is the difference between $A(x, y)$ minus the expected probability of an edge between x and y.

In his article, Newman deals with graphs considered as "scale free", the probability of an edge being proportional to the degrees of both ends. For this model, he establishes

$$B(x, y) = A(x, y) - \frac{Dg(x)Dg(y)}{2m}.$$

The aim is to maximize the function

$$F = \sum_{k=1,..p} \sum_{(x,y)\in P_k} B(x, y)$$

over nested partitions established by a hierarchical clustering process. The proposed algorithm is a divisive method starting from a single class and subdividing all the classes with more than one element. The subdivision of a class is made, while F increases, according to the spectral decomposition of matrix B. The resulting classes depend on the components of the eigenvector associated to the largest eigenvalue (whatever is its sign). Each subdivision is in three steps:

- the adjacency matrix, the degrees of internal edges of this class and the B matrix are established;
- the principal eigenvector is computed;
- the current class is subdivided in two subclasses corresponding to the positive and the negative component values.

In our simulations (*cf.* Sect. 6), we select random graphs according to the mixture Erdös-Reyni model; the probability of an edge is uniformly equal to $\frac{2m}{n(n-1)}$ and does not depend on the degree values. We have compared the original function B and

$$B'(x, y) = A(x, y) - \frac{2m}{n(n-1)}.$$

As the results are clearly better with formula B', for our random graphs, we keep this latter to apply the same algorithm. Here, the divisive process is stopped when the number of required classes is reached.

Example 4. At the first iteration the following principal eigenvector is computed, giving a first subdivision: {1, 2, 3, 4, 5, 6, 8, 9}, {10, 11, 12, 13, 14, 15}.

Largest eigenvalue –3.109 Eigenvector

1	10	11	12	13	14	15	2	3	4	5	6	7	8	9
0.14	-0.77	-1.00	-0.87	-0.99	-0.54	-0.57	0.52	0.48	0.17	0.58	0.72	0.78	0.82	0.46

For the second iteration, we subdivide class 1 (the largest one) computing the main eigenvector of matrix B' which gives the natural partition:

```
      1     2     3     4     5     6     7     8     9
1 : -0.44  0.56  0.56  0.56 -0.44 -0.44 -0.44 -0.44 -0.44
2 :  0.56 -0.44  0.56  0.56 -0.44 -0.44  0.56 -0.44 -0.44
3 :  0.56  0.56 -0.44  0.56  0.56 -0.44 -0.44 -0.44 -0.44
4 :  0.56  0.56  0.56 -0.44 -0.44 -0.44 -0.44 -0.44 -0.44
5 : -0.44 -0.44  0.56 -0.44 -0.44  0.56 -0.44  0.56  0.56
6 : -0.44 -0.44 -0.44 -0.44  0.56 -0.44  0.56  0.56 -0.44
7 : -0.44  0.56 -0.44 -0.44 -0.44  0.56 -0.44  0.56  0.56
8 : -0.44 -0.44 -0.44 -0.44  0.56  0.56  0.56 -0.44  0.56
9 : -0.44 -0.44 -0.44 -0.44  0.56 -0.44  0.56  0.56 -0.44

Largest eigenvalue 2.759
Eigenvector
  1     2     3     4    5    6    7    8    9
-1.00 -0.82 -0.82 -1.00 0.59 0.80 0.59 1.00 0.80
```

5. A classical optimization method

In the PartOpt algorithm, we try to minimize a criterion over the set $\mathcal{P}_p$ of all the partitions of X in p classes. Our criterion is defined as an inertia function $\Im : \mathcal{P}_p \to \mathbb{R}$ evaluated from the Czekanovski-Dice distance. Given a partition, $P = (P_1, P_2, ...P_p)$ in which $P(x)$ is the class number of x, $\Im(P)$ is the sum, over all the elements, of the square of the average distance to the others elements in its class:

$$\Im(P) = \sum_{x \in X} \left(\frac{\sum_{y \in P(x)} D(x,y)}{|P(x)| - 1} \right)^2 .$$

Another valuable distance on graphs has been proposed by Pons and Latapy [22]. It is defined as the euclidean distance between transition probability vectors. These later are established after random walks in the graph as in the MCL procedure, but there is no *inflation* operator. For vertex x the composants are the probabilities to reach any y from x after a small number of steps. We compare it to the Czekanovski-Dice distance but we get not so good results whatever was the length of the walk.

The optimization procedure is a simple Tabu-search heuristic. At each step, one element is transferred into another class. For each element, which is not a singleton, a new class is selected at random and the resulting variation of $\Im$ is computed. The moving element is determined as the one which minimizes the variation, even if it is positive, and this element is fixed within this class during a few steps (the length of the tabou list). Each iteration requires $O(n^2)$ operations. After n unsuccessful trials to decrease $\Im$, the best partition is kept. We have tested that this $\Im$ criterion gives results equivalent to those of other criteria as the percentage

of internal edges. This very simple algorithm surpasses several others optimization strategies, for different types of data, according to first simulations [15].

To apply this method, an initial partition and the number of classes are required. For the initial partition, the class number of each element is randomly selected between 1 and p. When the number of classes is to determine, sub optimal partitions for $p = 2, 3, ...$ etc. are established until the modularity function Q, introduced by Newman and Girwan [20], decreases. Its formula is

$$Q = \sum_{k=1,..p} e_k - a_k^2$$

where e_k is the percentage of edges within class P_k ($e_k = \frac{|E(P_k)|}{m}$) and a_k is the percentage of edges having at least one end in P_k, the inter-class edges contributing for 1/2 to each class ($a_k = \frac{|E(P_k)|+1/2|\Gamma(P_k)|}{m}$). This quantity Q measures the fraction of intra-class edges minus the expected value if the m edges were put at random. It quantifies how much the partition fit to the graph and it is often used to determine the number of natural clusters [12,20,22]. As we shall see in the simulation section, it works perfectly well, much better than the famous Calinski-Harabash rule [17] or other rules based on criterion variations.

The PartOpt algorithm

(1) From $\Gamma = (X, E)$, compute the Czekanovski-Dice distance on X.
(2) $p = 1$; $BestQ = 0$.
(3) While (Q increases)
   - select a random partition with $p + 1$ classes;
   - apply the Tabu-search procedure to get a sub-optimal partition P;
   - evaluate the Modularity function $Q(P)$;
   - if ($Q(P) > BestQ$) $\{p = p + 1$, $P* = P$, $BestQ = Q(P)\}$.

(4) The retained partition is $P*$ obtained at the previous step and containing p classes.

Example 5. For the graph test, the successive iterations give:

```
Initial value of the criterion : 7.61
Nb. of classes 2 : {1,2,3,4,5,6,7,8,9 | 10,11,12,13,14,15}
BestCrit 4.12 Mod = .347
Nb. of classes 3 : {1,2,3,4 | 5,6,7,8,9 | 10,11,12,13,14,15}
BestCrit 1.72 Mod = .458
Nb. of classes 4 : {1,2,3,4 | 5,6,7,8,9 | 10,11,14 | 12,13,15}
BestCrit 1.22 Mod = .404
```

The algorithm stops because the modularity decreases and the natural partition in 3 classes is established.

6. Validation by simulation

In order to evaluate the ability of these methods to detect high density zones, we try to recover such classes initially introduced in a graph. We first develop a generator of random graphs in which there are dense clusters having more edges within the classes – the internal edges – than between elements in separate classes – the external ones.

6.1. A Generator of random graphs

Each vertex belongs to one class which is a high density zone compared to the whole graph. Our generator of random graphs depends on four parameters:

- n: the number of vertices;
- p: the number of initial classes in the graph;
- p_i: the internal edge probability;
- p_e: the external edge probability.

In order to get such a random graph, we begin with a random partition of X in p classes, denoted $P = (P_1, .., P_p)$, having a variable number of elements per class, but which are balanced in the average. Next, for each pair of elements, we select at random a real number between 0 and 1, and we add the corresponding edge if and only if this number is lower than or equal to p_i (resp. p_e) when the two elements are in identical (resp. different) classes. This is the Erdös-Reyni procedure applied within the classes and also to G for the external edges.

Each clustering algorithm returns a partition $Q = (Q_1, ..., Q_q)$. The number of classes can be fixed (and $q = p$) or it can be freely determined by the algorithm, possibly giving $q \neq p$. Let $q_j = |Q_j|$ and $Q(x)$ be the class number of x.

6.2. Quality of the classes compared to the initial partition

In order to compare the five algorithms described above, we evaluate how far is Q from P using four criteria which do not impose to have $p = q$. The two first ones are not symmetrical; they compare the classes of Q to those of P and the results would not be the same if P was compared to Q. The third one is the classical *Rand index corrected for chance* (Hubert and Arabie, 1985) and the fourth is an editing distance between partitions.

- τ_e: the percentage of elements which belong to their *corresponding* class in P. We first map the classes of Q onto those of P evaluating $n_{i,j} = |P_i \bigcap Q_j|$. We define the class in P *corresponding* to Q_j, denoted $P_{\sigma(j)}$, as the one that contains the greatest number of elements of Q_j; that is $n_{\sigma(j),j} \geq n_{i,j}$ for all i from 1 to p. So defined, σ is not a permutation, since two classes in Q can share the same corresponding class, for instance when a class in P is subdivided into two classes in Q

$$\tau_e = \sum_{j=1,..q} \frac{|Q_j \bigcap P_{\sigma(j)}|}{|Q_j|}.$$

- τ_p: the percentage of joined pairs in Q that are also joined in P. Let $\pi(P)$ be the set of joined pairs in partition P

$$\tau_p = \frac{|\pi(P) \cap \pi(Q)|}{|\pi(Q)|}.$$

- The Rand index corrected for chance, denoted R_c, is based on three values: r is the number of common joined pairs in P and Q ($r = |\pi(P) \cap \pi(Q)|$), the expected value $Exp(r)$ over the partitions of the same type as P and Q and the maximum value $\mathrm{Max}(r)$. The authors adopt the formula:

$$R_c(P, Q) = \frac{r - Exp(r)}{\mathrm{Max}(r) - Exp(r)}$$

  with $Exp(r) = \frac{|\pi(P)| \times |\pi(Q)|}{n(n-1)/2}$ and $\mathrm{Max}(r) = \frac{1}{2}(|\pi(P)| + |\pi(Q)|)$. Note that this index can take negative values corresponding to partitions that are independent.
- τ_t is the *transfer distance* value divided by n. This distance counts the minimum number of transfers, of one element from one class to another, possibly a new class, that are necessary to transform P into Q. This distance between any two partitions has been proposed by Regnier [23] and Day [10] and recently studied (and bounded according to the class cardinality) by Charon *et al.* [8]. It is well adapted to very close partitions because a small number of transfers reveals partitions that are practically identical. Its value is obtained realizing a matching of the classes of P onto those of Q which maximizes the number of common elements in matched classes.

6.3. Results

There are two series of tests: the first one corresponds to methods in which the number of classes is fixed. In that case, the pruning, the spectral and the optimisation methods are compared. When the number of classes is free, the density, the Markov and the optimisation methods are compared. For both series, in Table 4, the external density is fixed to $p_e = .1$ and the internal density (percentage of edges) decreases, until the computed partitions becomes too far from the initial ones. In Table 5, it is the converse; the internal edge probability is fixed to $p_i = .5$ and the external one increases.

These figures are average values obtained from 200 random graphs of 100 vertices distributed in 3 classes, that are only balanced in the average; the degree values typically span from 3 to more that 10. In the following tables, the rows correspond to the algorithms and the columns to the four criteria introduced before.

The number of classes is fixed

When $p_i = .5$ and $p_e = .1$ all these methods are very satisfying and the initial classes are perfectly recovered (one or two transfers). But when the gap between

TABLE 4. The internal edge probability go decreasing.

	$p_i = .4, p_e = .1$				$p_i = .35, p_e = .1$				$p_i = .3, p_e = .1$			
	τ_e	τ_p	R_c	τ_t	τ_e	τ_p	R_c	τ_t	τ_e	τ_p	R_c	τ_t
Pruning	.96	.94	.92	.03	.91	.85	.80	.09	.80	.70	.60	.20
Spectral	.95	.91	.88	.05	.93	.86	.80	.07	.88	.78	.69	.12
Optimization	.99	.98	.97	.01	.98	.96	.95	.02	.94	.89	.84	.06

TABLE 5. External edge probability increases.

	$p_i = .5, p_e = .15$				$p_i = .5, p_e = .2$				$p_i = .5, p_e = .25$			
	τ_e	τ_p	R_c	τ_t	τ_e	τ_p	R_c	τ_t	τ_e	τ_p	R_c	τ_t
Pruning	.98	.96	.95	.02	.93	.87	.84	.07	.85	.75	.67	.15
Spectral	.96	.91	.89	.04	.92	.85	.80	.08	.89	.78	.69	.13
Optimization	.99	.99	.98	.01	.98	.96	.95	.02	.94	.89	.83	.06

TABLE 6. Internal edge probability decreases.

	$p_i = .4, p_e = .1$				$p_i = .35, p_e = .1$				$p_i = .3, p_e = .1$			
	τ_e	τ_p	R_c	τ_t	τ_e	τ_p	R_c	τ_t	τ_e	τ_p	R_c	τ_t
Density	.83	.76	.56	.25	.79	.70	.45	.33	.71	.59	.29	.44
Markov	.93	.87	.82	.08	.83	.73	.58	.22	.60	.47	.22	.43
Optimization	.99	.98	.97	.01	.97	.95	.93	.03	.95	.90	.85	.06

these probabilities decreases, the optimization method clearly becomes the best one. It is obvious for the harder cases, when $p_i = .5$ and $p_e = .25$ or for $p_i = .3$ and $p_e = .1$.

The number of classes is free

Once again, the optimisation procedure is the best of them three. For $p_i = .5$ and $p_e = .1$ the MCL procedure recovers practically every time the correct number of classes, with $r = 2$. But when the gap between p_i and p_e decreases, the r-value must be adjusted with many trials to get an average number of classes close to 3. For instance, for $p_i = .3, p_e = .1$, with $r = 2$ we get 8.4 elements per class in the average, but with $r = 1.95$ we get only 3.6. In fact r should be adapted to each graph, but we only do in the average. For Table 6, the retained r values are respectively 2.1, 2.05 and 1.95. For Table 7, with $r = 2$, all the vertices would be joined together into a single class as soon as $p_e \geq .15$; here again, after many trials, we keep respectively 2.4, 2.55, and 2.75. The density method gives also poor results even if the number of classes remains reasonable. Remarkably, the optimization one has the same performance as when the number of classes is fixed, because this latter is practically always recovered. But we must admit that it is so because there is a small number of classes with a few tens of elements; with

TABLE 7. External edge probability increases.

	$p_i = .5, p_e = .15$				$p_i = .5, p_e = .20$				$p_i = .5, p_e = .25$			
	τ_e	τ_p	R_c	τ_t	τ_e	τ_p	R_c	τ_t	τ_e	τ_p	R_c	τ_t
Density	.73	.64	.44	.29	.58	.48	.21	.42	.48	.40	.08	.52
Markov	.86	.80	.69	.16	.64	.51	.30	.38	.50	.39	.10	.52
Optimization	.99	.98	.97	.01	.98	.95	.95	.03	.93	.87	.82	.07

TABLE 8. Results of the PartOpt method applied to sparse graphs of 200 vertices.

			♯ of classes fixed				♯ of classes free			
p	p_i	p_e	τ_e	τ_p	R_c	τ_t	τ_e	τ_p	R_c	τ_t
4	.15	.03	.94	.89	.86	.06	.95	.90	.87	.06
4	.10	.01	.95	.90	.87	.05	.94	.90	.86	.06
5	.15	.03	.87	.78	.73	.13	.86	.76	.70	.14
5	.10	.01	.86	.77	.72	.14	.86	.76	.70	.15

200 vertices in 10 classes in the initial graph, it would not be the same, except for $p_i = .5$ and $p_e = .1$, making obvious denses clusters.

The PartOpt performances remain satisfying when classes are unbalanced, and when n increases or when graphs becomes more sparse. In this last table, $n = 200$ and the number of classes is fixed to 4 or 5. The internal and external probabilities lead to graphs with average density between .06 and .03. The values of the same criteria tend to prove that the optimization algorithm can be applied to sparse graphs.

7. CONCLUSION

According to Tables 4 to 8, graph partitioning could appear as an easy problem, since the five methods based on different principles provide acceptable results in the average, except MCL when the density is high. It is time to admit that this simulation process is driven to get a clear preference. It would not be so clear if, for instance, the initial classes contain less than a few tens of vertices, if the average density was lower than .2 or if the number of classes was larger than 10. In any of these cases all the algorithms fail to recover the graph structure, beginning with the correct number of classes, and the comparison between them would be hazardous.

Finally, new and promising ideas fail to recover existing classes in graphs, when the contrast between high density zones is not clear. A simple good old stochastic optimization procedure gives the best results without an extensive study of its parameters (the number of initial partitions, or the number of iterations without improvements, here equal to n, and the length of tabu lists, always equal to 5). The computing time is also satisfying since a sample of 100 graphs of 100 vertices is analyzed in 15" and for graphs of 200 vertices, it takes around 100" (when the number of classes is free) using an ordinary desk computer.

References

[1] C.J. Alpert and A. Kang, Recent direction in netlist partitioning: a survey, *Integration. VLSI J.* **19 (1-2)** (1995) 1–81.

[2] G.D. Bader and C.W. Hogue, An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics* **4 (2)** (2003) 27.

[3] L. Barabasi, The large-scale organization of metabolic networks. *Nature* **407** (2000) 651–654.

[4] V. Batagelj and M. Mrvar, Partitioning approach to visualisation of large graphs, *Lect. Notes Comput. Sci.* 1731, Springer (1999) 90–97.

[5] V. Batagelj and M. Zaveršnik, An $O(m)$ algorithm for cores decomposition of networks (2001).

[6] S. Brohée and J. van Helden, Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics* **7** (2006) 488.

[7] C. Brun, C. Herrmann and A. Guénoche, Clustering proteins from interaction networks for the prediction of cellular functions. *BMC Bioinformatics* **5** (2004) 95.

[8] I. Charon, L. Denoeud, A. Guénoche, and O. Hudry, Comparing partitions by element transfert. *J. Classif.* **23 (1)** (2006) 103–121.

[9] T. Colombo, A. Guénoche, and Y. Quentin, Looking for high density areas in graph: Application to orthologous genes, Actes des Journées Informatiques de Metz, 2003, pp. 203–212.

[10] W. Day, The complexity of computing metric distances between partitions. *Math. Soc. Sci.* **1** (1981) 269–287.

[11] S. Van Dongen, *Graph Clustering by Flow Simulation.* Ph.D. Thesis, University of Utrecht (2000).

[12] J. Duch and A. Arenas, Community detection in complex networks using Extremal Optimization, `arXiv:cond-mat/0501368` (2005) 4 p.

[13] A.J. Enright, S. van Dongen and L.A. Ouzounis, An efficient algorithm for large-scale detection of protein families. *Nucleic Acids Res.* **30** (2002) 1575–1584.

[14] M. Girvan and M.E.J. Newman, Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA* **99** (2002) 7821–7826.

[15] A. Guénoche, Partitions optimisées selon différents critères; Evaluation et comparaison. *Math. Sci. Hum.* **161** (2003) 41–58.

[16] A. Guénoche, Clustering by vertex density in a graph, in *P*roceedings of IFCS congress. *Classification, Clustering and Data Mining Applications*, edited by D. Banks et al., Springer (2004) 15–24.

[17] G.W. Milligan and M.C. Cooper, An examination of procedures for determining the number of clusters in a data set. *Psychometrica* **50** (1985) 159–179.

[18] J. Moody, Identifying dense clusters in large networks. *Social Networks* **23** (2001) 261–283.

[19] M.E.J. Newman, Scientific Collaboration Networks: Shortest paths, weighted networks and centrality. *Phys. Rev.* (2001) 64.

[20] M.E.J Newman and M. Girvan, Finding and evaluating community structure in networks. *Phys. Rev. E* **69** (2004) 026113.

[21] M.E.J. Newman, Modularity and community structure in networks. `arXiv:physics/0602124v1`, (2006) 7 p.

[22] P. Pons and M. Latapy, Computing communities in large networks using random walks. *J. Graph Algorithms Appl.* **10 (2)**, (2006) 191–218.

[23] S. Régnier, Sur quelques aspects mathématiques des problèmes de classification automatique. *ICC Bulletin* (1964).

[24] J. Rougemont and P. Hingamp, DNA microarray data and contextual analysis of correlation graphs. *BMC Bioinformatics* **4** (2003) 15.

[25] S.B. Seidman, Network structure and minimum degree. *Social Networks* **5** (1983) 269–287.

[26] D. Wishart, Mode analysis: generalization of nearest neighbor which reduces chaining effects, in *Numerical taxonomy*, Academic Press (1976) 282–311.