ModClust: a Cytoscape plugin for modularity-based clustering of networks

Laurent Tichit, Philippe Gambette, Alain Guénoche

ModClust: a Cytoscape plugin for modularity-based clustering of networks

Laurent Tichit [a,b], **Philippe Gambette** [b,c] **and Alain Guénoche** [b]

[a] *Université d'Aix-Marseille*
[b] *Institut de Mathématiques de Luminy*
[c] *Université de Marne-la-Vallée*

ABSTRACT. *Large networks such as protein-protein interaction networks are usually extremely difficult to understand as a whole. We developed ModClust, a Cytoscape plugin for modularity-based clustering of large networks. The aim of this plugin is first to establish classes of high density edges. It also allows to understand the relations between these classes, and how they are assembled within the whole graph. It can be used to predict new protein functions. It implements two novel algorithms:* FT *and* TFit*. Their results are compared both on random graphs and on benchmarks where the optimal partition is known.*

RÉSUMÉ. *Les grands graphes, comme les réseaux d'interaction protéine-protéine, sont d'une manière générale difficiles à analyser. Nous avons développé un plugin pour le logiciel Cytoscape, appelé ModClust, effectuant du partitionnement de graphes par optimisation de la modularité. L'objectif de ce plugin est de comprendre quelles sont les relations entre classes et comment ces dernières sont assemblées dans le graphe. Il nous aide finalement à prédire de nouvelles fonctions protéiques. Deux nouveaux algorithmes,* FT *et* TFit*, sont implémentés. Leurs résultats sont comparés sur des graphes aléatoires et sur des benchmarks dont on connait les partitions optimales.*

KEYWORDS: *Graph partitioning, protein-protein interaction network, modularity*

MOTS-CLÉS : *Graphe, partitionnement, réseau d'interactions protéine-protéine, modularité*

1. Introduction

As soon as their number of vertices becomes important, large graphs such as protein-protein interaction (PPI) networks become tremendously difficult to understand as a whole. To get a clear visualization of such large graphs, it is necessary to cluster them, to draw classes separately, and to emphasize the links between them. In the same time, drawing and clustering are two unavoidable and intricate tools to provide understanding of the relation expressed by the graph. At a more specific level, in a protein-protein interaction network [Schwikowski *et al*., 2000, Xenarios *et al*., 2000], graph vertices are proteins and an edge exists between two proteins if they somehow interact (and thus participate in the same function). Efficient clustering of PPI networks can lead to the prediction of novel protein functions.

The graph partitioning problem has a long history we don't detail here. On a practical point of view, it has recently become highly relevant in at least three domains:

– biological problems modeled by graphs [Schwikowski *et al*., 2000],

– the study of large networks like the Web [Moody, 2001],

– the definition of *communities* in social networks [Newman and Girvan, 2004].

Every time, the aim is to gather together vertices sharing a large number of edges, making some *high density zones*, compared to the percentage observed in the whole graph. In the first part, we introduce two algorithms, *FT* and *TFit*. We compare them to the optimal solution when possible, and to other heuristics, on graphs commonly used as benchmark. Then, we present the software infrastructure of our Cytoscape [Shannon *et al*., 2003] plugin, its features and capabilities. Finally, we apply them to a biological graph (interactions between proteins in *Plasmodium falciparum*) and compare the two obtained partitions.

The novel heuristics we have implemented try to optimize the modularity criterion. Let $G = (X, E)$ be an undirected, unweighted graph with $|X| = n$ and $|E| = m$. Let $A(x, y) = 1 \iff (x, y)$ is an edge (0 otherwise) and let D_x be the degree of x. The aim of these algorithms is to optimize an integer modularity function equivalent to Newman's original definition of modularity. The problem becomes a clique partitioning problem on a complete graph where each pair (x, y) is weighted by

$$w(x, y) = 2m \times A(x, y) - D_x \times D_y.$$

Let $\pi = (X_1, \ldots, X_p)$ be a partition of X in p classes and $W(X_k) = \sum_{x,y \in X_k} w(x, y)$ the modularity of class X_k. The modularity function to optimize over the set of all the partitions of X is

$$W(\pi) = \sum_{k=1}^{p} W(X_k).$$

Both algorithms, *FT* and *TFit*, output a strict partition (disjoint classes covering the whole set of vertices).

2. Modularity-based Fusion-Transfer method: FT

As the clique partitioning problem is NP-complete, several approximation methods have been developed, the first one being the *Transfer method* proposed by Régnier [Régnier, 1965]. Starting from a random partition, it consists in moving one vertex from its class to another one, as long as the modularity criterion increases. It is a simple *hill-climbing* method which leads to a local maximum of the scoring function. We present a heuristic to optimize W_π which gives excellent results. It is based on the average linkage ascending method and on the Transfer method, followed by a stochastic optimization procedure. This algorithm is divided into three steps:

1) The first one, called *Fusion*, follows a bottom-up approach. It starts from the atomic partition P_0, and at each step, merges the two classes that maximize the score of the resulting partition. These classes are the ones for which the sum of the inter-class edge weights is maximum. The process stops when no further merge leads to an increase of the score. This defines p as well as a partition $\pi = (X_1, \ldots, X_p)$ such that every partition π_{ij} resulting from the union of X_i and X_j has a lower score: $W(\pi_{ij}) < W(\pi)$.

2) In the second phase, called *Transfer*, we first compute the *contribution* of each vertex x_i to any class X_k. Let

$$K(x_i, k) = \sum_{x_j \in X_k} w(x_i, x_j).$$

If $x_i \in X_k$, $K(x_i, k)$ is the contribution of x_i to its class and to the current partition. Otherwise, this value corresponds to a possible allocation to another class $X_{k'}$. The difference $K(x_i, k') - K(x_i, k)$ is the score variation resulting from the transfer of x_i from class X_k to class $X_{k'}$. Our procedure consists in moving, at each step, the vertex that maximizes the score increase. It can be moved either to another class or to a new class, making a singleton. This deterministic algorithm returns a partition π.

3) The third step consists in a stochastic optimization procedure. For each trial, we start from a random partition derived from π by swapping elements between classes. Then we apply the transfer procedure until we get π' which can improve the W criterion. There are two parameters to define: the maximum number of swaps ($SwapMax$ fixed to $2 \times N/NbClas$), and the maximum number of consecutive trials without modularity improvement ($NbTrials$ fixed to N and bounded by 500).

3. The Iterated Transfer-Fusion Method ($TFit$)

This previous heuristic corresponds mostly to an agglomerative hierarchical clustering methods, where cluster fusion is the basic operation. In the fast and popular Louvain method [Blondel *et al.*, 2008], the main operation is a transfer of elements or clusters. Each step of cluster fusion consists in creating clusters of clusters, and transferring the clusters of vertices, from one cluster of clusters to another, as long as modularity increases. We call this operation "cluster transfer". Once this step is over, clusters of vertices belonging to the same cluster of clusters are merged.

Algorithm 1 FT algorithm

Require: $G = (V, E)$: graph
  [**Fusion procedure**]
  Starting from $P0$
  Compute the weight of each pair of singletons ($w(i, j)$)
  while score increases **do**
    merge the two classes giving the maximum score
    update fusion costs between new class and every other
  end while
  [**Transfer procedure**]
  For every class, compute the contribution of each element to each class
  while exists an element with negative or non maximum weight **do**
    transfer it to the class where its contribution is maximum, or make it a singleton
    update weights in both modified classes
  end while
  [**Stochastic optimization procedure**]
  $trials \leftarrow 0$
  while $trials < NbTrials$ **do**
    Let $swap$ be a random integer between 1 and $SwapMax$
    Starting from π, exchange $swap$ randomly chosen elements
    call the Transfer procedure which returns π
    if $W(\pi') > W(\pi)$ **then**
      $\pi \leftarrow \pi'; trials \leftarrow 0;$
    else
      $trials \leftarrow trials + 1$
    end if
  end while
  return Partition π;

The function which decides which pair will be fusioned next is called a "merge prioritizer" in a recent comparative study of heuristics for modularity optimization [Noack and Rotta, 2009]. In the Louvain method [Blondel *et al.*, 2008] as well as in $TFit$, we use no "merge prioritizer" (those may create unbalanced clusters), and just consider in turn all clusters or vertices to transfer. Noack and Rotta also mention some possible post-processings, called "refinements", based on vertex transfers, to improve modularity. They claim that this step increases the running time. However, for some graphs like protein-protein interaction graphs which have a few thousands of vertices, we are not limited by this running time issue, and we can use the "fast greedy" version of this improvement heuristic, which consists in transferring each vertex to the cluster with best modularity improvement, if any. Furthermore, we do not only apply it at the end of the algorithm, but before each cluster fusion step, which explains the name of our algorithm. Briefly speaking, $TFit$ is a multi-level algorithm in which an element transfer procedure has been inserted at each level change.

More formally, algorithm $TFit$ is described in Fig. 2. Note that as it is explained for FT, it is easy to compute the potential modularity increase for each vertex transfer, and to add a final modularity stochastic optimization procedure.

4. Performance of FT and $TFit$

4.1. *On Benchmark Graphs*

A set of graphs corresponding to real data have been used as common benchmarks in many articles which compare graph clustering algorithms through modularity opti-

Algorithm 2 $TFit$ algorithm. A call to **Transfer**(v,P_i,P) deletes the element v (either a vertex or a cluster) from its cluster in P, adds it to the cluster $P_i \in P$, and returns P. A call to **Fusion**($P' = \{P'_i\}$) returns $P = \{\bigcup_{C_i \in P'_i} C_i\}$.

Require: $G = (V, E)$: graph
  $P \leftarrow \{\{x\}, x \in V\}$; $mod \leftarrow 0$; $currentmod \leftarrow mod$; $continue \leftarrow$ true;
  while $continue$ **do**
    while $\exists v \in V, C_i \in P \cup \{\emptyset\}$ such that
    modularity(G,P) < modularity(G,**Transfer**(v,C_i,P)) **do**
      $C_i \leftarrow$ argmax $\Big($ modularity(G,**Transfer**(v,C_i,P)) - modularity(G,P)$\Big)$
      $P \leftarrow$ **Transfer**(v,C_i,P)
      $currentmod \leftarrow$ modularity(G,P)
    end while
    $P' \leftarrow \Big\{\{C_i\}\Big\}$;
    while $\exists C_i \in P, P'_j \in P' \cup \{\emptyset\}$ such that
    modularity(G,P') < modularity $\Big(G$,**Fusion**(**Transfer**(C_i,P'_j,P'))$\Big)$ **do**
      $P'_j \leftarrow$ argmax $\Big[$modularity $\Big(G$,**Fusion**(**Transfer**(C_i,P'_j,P'))$\Big)$ - modularity $\Big($G,**Fusion**(P')$\Big)\Big]$
      $P' \leftarrow$ **Transfer**(C_i,P'_j,P')
      $currentmod \leftarrow$ modularity $\Big($G,**Fusion**(P')$\Big)$
    end while
    $P \leftarrow$ **Fusion**(P')
    if $currentmod \leq mod$ **then**
      $continue \leftarrow$ false
    else
      $continue \leftarrow$ true
      $mod \leftarrow currentmod$
    end if
  end while
  return Partition P;

mization. For some of them (with at most a few hundred vertices), an optimal solution was computed by Integer Linear Programming [Aloise *et al.*, 2010]. We compared our heuristic to the Louvain method and to the best modularity value obtained by the set of heuristics described by Noack & Rotta. As shown in Table 1, our methods give better results than the Louvain method, and sometimes outperforms the ten Noack & Rotta heuristics. Note that if we compare $TFit$ and FT with each of those heuristics, the modularity found is equal or better in at least 5 of the 8 benchmark graphs.

4.2. *Comparison on random graphs*

We have also developed a test program comparing the results of these two graph partitioning algorithms on random graphs. We first define a reference partition P_r of the vertices. The number of classes (whose sizes are more or less balanced) is given as user input. Then, the edges are drawn according to two different probabilities which give densities (D_i, D_e) for intra (internal) or inter-class (external) edges. FT and $TFit$ respectively compute partitions π_1 and π_2; in Table 2, we indicate the average

graph	n	m	Opt	Louvain	N-R	FT	TFit
Dolphins	62	159	.5285	.5185	.5276	**.5285**	.5268
polBooks	105	441	.5272	.5266	**.5272**	.5221	.5269
afootball	115	613	.6046	**.6046**	.6045	.6032	**.6046**
A01	249	635	.6329	.6145	.6293	**.6310**	.6294
USAir97	332	2126	.3682	.3541	.3678	**.3682**	.3612
netscience	379	914	.8486	.8475	.8474	.8474	**.8477**
s388	512	819	.8194	.7962	.8143	.8122	**.8154**
emails	1133	5452		.5438	**.5816**	.5556	.5747

Table 1. *Comparison of FT and $TFit$ on benchmark graphs, with stochastic optimization.*

N / # classes	D_i/D_e edges	FT # classes	TFit # classes	Mod. gain	Rand gain	Mod. decr.	Rand decr.
200 / 5	.30 / .10	5.7	**4.9**	0.04	**0.08**	0.17	**0.26**
	.20 / .5	6.8	**5.4**	0.05	**0.11**	0.19	**0.28**
	.10 / .1	7.8	**6.7**	0.04	**0.12**	0.10	0.12
500 / 5	.30 / .10	5.0	5.0	0.00	0.00	0.02	0.02
	.20 / .5	5.0	5.0	0.00	0.00	0.01	0.00
	.10 / .1	5.0	5.0	0.00	0.00	0.03	0.01
300 / 10	.30 / .10	**8.0**	7.3	0.03	0.01	**0.32**	**0.45**
	.20 / .5	**8.8**	8.0	0.04	0.00	0.22	**0.46**
	.10 / .1	11.0	**10.4**	0.04	**0.06**	0.03	0.18

Table 2. *Comparison of FT and TFit on random graphs*

number of computed classes. Once the partition has been computed, the proximity between it and P_r is measured using the corrected Rand index [Hubert and Arabie, 1985].

Then, we compute the *modularity gain* of $TFit$ with regard to FT ($Mod.\ gain = (W(\pi_2) - W(\pi_1))/W(\pi_1)$) and the *modularity decrease*, that is the rate of tests where FT is better than $TFit$, modularity-wise. We compute as well the *rand gain* of $TFit$ with regard to FT ($Rand\ gain = Rand(\pi_2, Pr) - Rand(\pi_1, Pr)$) and the *rand decrease*.

Evaluating the benchmarks, we can conclude that $TFit$ is in average better than FT, modularity-wise, especially on low density graphs. $TFit$ tends to find less classes (hence bigger classes) than FT. But in 20 % of the tests, FT achieves better results, modularity-wise. If P_r already contains huge classes, both algorithms succeed in finding them. In these cases, the benchmark results fail showing any difference between them.

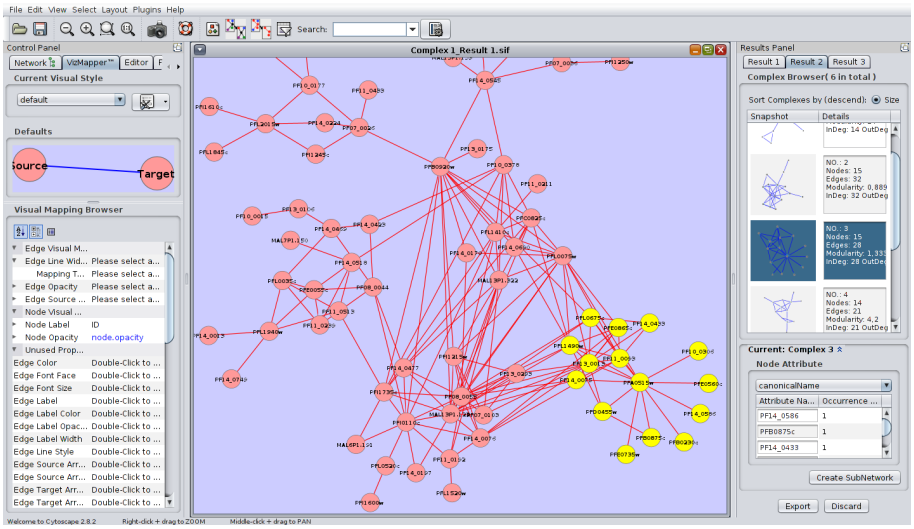

Figure 1. *Clustering of Plasmodium Falciparum PPI (1355 nodes, 12253 edges) using FT algorithm. After two passes of FT, this figure shows the class of 15 proteins working with kinase PFA0515w putatively involved in the phosphatidylinositol metabolic process.*

5. Plugin features

We have implemented both algorithms into a Cytoscape plugin based on ClusterViz [Chen *et al.*, 2010]. Beside the abilities of this plugin (free labels for nodes, automatic visualization and layout of the classes, class export as a new graph, clustering of just a part of the graph), ModClust makes use of all the power of the Cytoscape platform [Shannon *et al.*, 2003].

Moreover, ModClust implements novel functionalities:

– Specific coloring for each class. We aim to color classes automatically, without any manual intervention. The color proposition could then be interactively corrected by user-provided colors if necessary. The number of colors should be equal to the degree of the quotient graph. The quotient graph is a graph whose vertices are the classes and where an edge exists if and only if there exists at least one inter-class edge in the original graph.

– Emphasis and selection of border/internal nodes of each class. In a class, an internal node is a node whose direct neighbors belong to the same class. A border node is also adjacent to at least another class. This functionality aims to make a distinction between interactions that should be robust and the ones that tend to be weaker.

– Pruning of border nodes. Once the distinction between the border nodes and the internal nodes has been done, it is possible to get rid of the border nodes in order to focus on the strongest interactions.

– Selection/extraction of several classes to build another entry graph. It is possible to focus on a sub-partition and start a new partitioning process on this new graph. This process can be done iteratively on smaller sub-networks.

– Emphasis of inter-class edges. These edges between two classes are often interesting in that they connect proteins which could have several potential functions.

Due to the Cytoscape/Java overhead, the plugins are able to deal with 5000 nodes for FT and 10000 for $TFit$ (the C-version of the algorithms work respectively on 10000 and 50000 nodes). The C source code is available on `http://bioinformatics.lif.univ-mrs.fr/GraphPartitioning/`. The current version of the Java plugin is available on request to the first author.

6. References

[Aloise *et al*., 2010] Aloise, D., Cafieri, S., Caporossi, G., Hansen, P., Perron, S. and Liberti, L., Column generation algorithms for exact modularity maximization in networks, *Physical Review E*, 82:046112, 2010.

[Blondel *et al*., 2008] Blondel, VD., Guillaume, J.-L., Lambiotte, R. and Lefebvre, E., Fast unfolding of community hierarchies in large networks, *Journal of Statistical Mechanics: Theory and Experiment*, P10008, 2008.

[Chen *et al*., 2010] Chen, G., Cai, J. and Wang, J., A Cytoscape plugin for biological network clustering and visualization, *ISCB SCS*, 6, 2010.

[Gambette and Guénoche, 2011] Gambette, P. and Guénoche, A., Bootstrap clustering for graph partitioning, *submitted*, 2011.

[Newman and Girvan, 2004] Newman, ME. and Girvan, M., Finding and evaluating community structure in networks, *Phys. Rev. E*, 69(2), 2004.

[Hubert and Arabie, 1985] Hubert, L. and Arabie, P., Comparing partitions, *Journal of Classification*, 2:193–198, 1985.

[Moody, 2001] Moody, J., Peer influence groups: Identifying dense clusters in large netwoks, *Social Networks*, 23(4):261–283, 2001.

[Noack and Rotta, 2009] Noack, A. and Rotta, R., Multi-level Algorithms for Modularity Clustering, *SEA 2009*, 257–268, 2009.

[Régnier, 1965] Régnier, S. Sur quelques aspects mathématiques des probl'èmes de classification automatique, *Mathématiques et Sciences humaines*, 82:13-29, 1983, reprint of I.C.C. Bulletin 4: 175–191, 1965.

[Schwikowski *et al*., 2000] Schwikowski, B., Uetz, P. and Fields, S., A network of protein-protein interactions in yeast, *Nature Biotechnology*, 18(12):1257–1261, 2000.

[Shannon *et al*., 2003] Shannon, P., Markiel, A., Ozier, O., Baliga, NS., Wang, JT., Ramage, D., Amin, N., Schwikowski, B. and Ideker, T., Cytoscape: A software environment for integrated models of biomolecular interaction networks, *Genome Research*, 13(11):2498–504, 2003.

[Xenarios *et al*., 2000] Xenarios, I., Rice, DW., Salwinski, L., Baron, MK., Marcotte, EM. and Eisenberg, D., DIP: the database of interacting proteins, *Nucleic Acids Res.*, 28(1):289–291, 2000.