Les Cahiers du GERAD ISSN: 0711–2440

A Visual Environment to Study and Find Communities in Networks

G. Caporossi
S. Perron

G–2011–55

October 2011

A Visual Environment to Study and Find Communities in Networks

Gilles Caporossi
Sylvain Perron

GERAD & HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada, H3T 2A7

gilles.caporossi@gerad.ca
sylvain.perron@gerad.ca

October 2011

Les Cahiers du GERAD

G–2011–55

Abstract

In the last ten years, finding communities in networks, has been the subject of intense studies. If modularity maximization is still one of the mostly studied criterion, some criticism tend to arise and researchers in that field are more and more studying various criterion for finding communities in networks. To facilitate the task of studying various or new criterion, *Constellation* was developed. The software does not only allow an interactive definition of new criterion, it also provides a graphical interface and an optimization module that are useful tools for the researcher. This paper describes *Constellation* and its use.

Résumé

Durant les dix dernières années, trouver des communautés dans des réseaux a été l'objet d'intenses études. Si la maximisation de la modularité est toujours un des critères les plus étudiés, certaines critiques tendent à s'élever et les chercheurs du domaine étudient de plus en plus d'autres critères. Pour faciliter le travail d'étudier de nouveaux critères variés, *Constellation* a été développé. Ce logiciel ne permet pas seulement une définition interactive de nouveaux critères, il comporte aussi une interface graphique et un module d'optimisation qui sont autant d'outils utiles au chercheur. Cet article décrit *Constellation* et son utilisation.

1 Introduction

In order to help researchers studying networks and communities in networks, *Constellation* was recently designed. The main purpose of *Constellation* is to provide a graphical environment in which networks may be studied. In order to better stimulate this research, the network may be displayed on the computer screen and its vertices may be moved either automatically or by the mean of the mouse. The graphical interface and the interactive use of *Constellation* is described in Section 2. From that last few years, the interest in finding communities in networks is ever increasing. Indeed, various algorithms and criterion are under study and one task of researchers in that field is to analyze new criterion, identify its assets and drawbacks. *Constellation* provides the researcher the ability to study new criterion by the mean of an interactive criterion definition whose syntax is described in Section 3. Still to help studying criterion, a generic optimization module is provided that allows reasonable performances for a wide variety of problems. The optimization module of *Constellation* is described in Section 4. Finally, a classical application example is provided in Section 5.

2 Interactive use of *Constellation*

A good approach when interactively studying a new graph is to start by asking the system to randomly place the vertices. According to the number of vertices and the density of the graph, the scale may then be adapted. The next step is to ask the system to propose a tentative community detection by defining the criterion under study as defined in Section 3. The criterion being defined, the user may run the optimization after setting the proper parameters as described in Section 4. After completing the optimization, the underlying structure of the graph may be exhibited in few steps. The first approach is to click the "separate clusters" button few times as to display the links between various clusters. The user then has the opportunity to arrange clusters interactively by first moving a vertex of the proper cluster (identified by its color) and then translating all other vertices of that same cluster accordingly by pressing the "m" key.

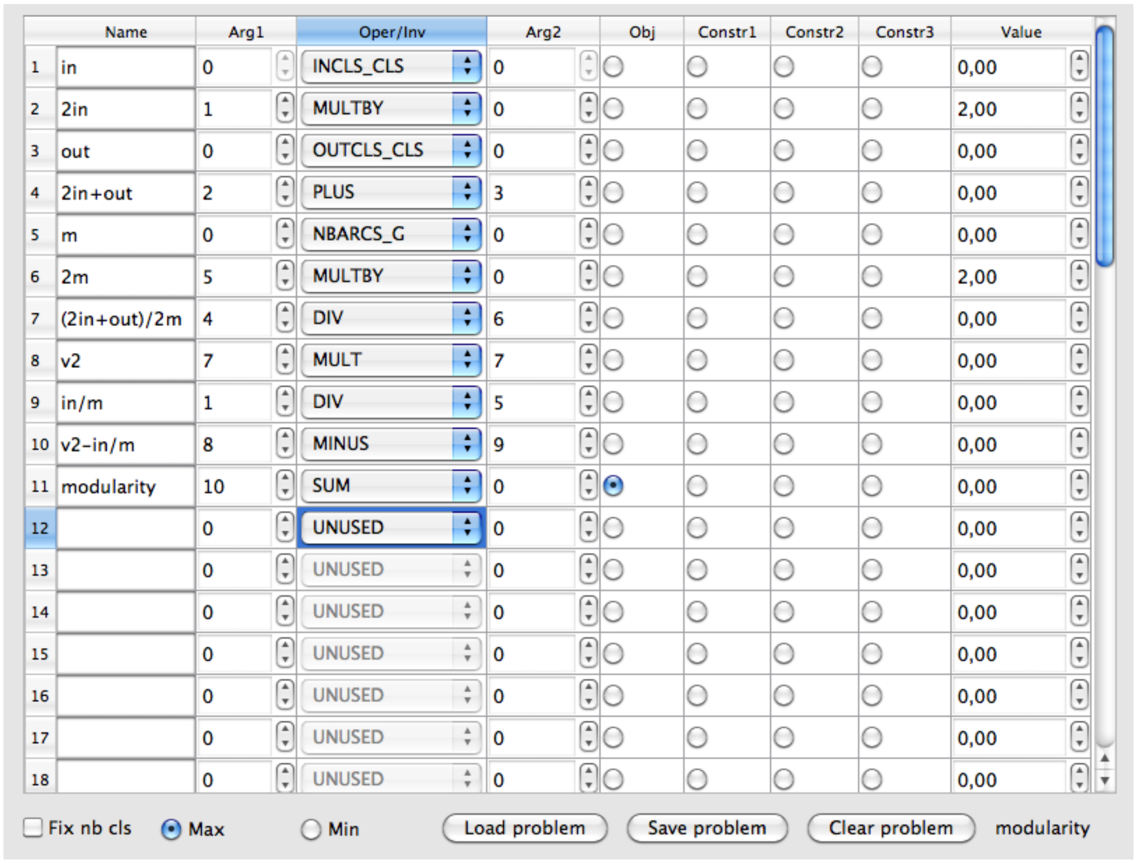
When opening a graph in *Constellation*, the user could see the graph on the right and some information characterizing it on the left. Among the available information, one can read the number of vertices, the number of edges, the diameter, the radius,the average distance, the minimum and maximum transmissions as well as the minimum and maximum degrees.

Through a tab, it is possible to adjust the size of vertices (to reflect their degree) and some position routines are also available to help identifying the structure found by the clustering algorithm.

By the mean of a graphical interface, it is possible to define the criterion to use for clustering vertices of the graph. As it implements a powerful heuristic that is robust enough to handle a wide variety of problems, one of the major use of *Constellation* is certainly finding clusters or communities in a graph.

3 Problem definition

Constellation was designed as a general toolbox for researchers willing to study communities in graphs. Instead of providing a series of clustering criterion, *Constellation* has a simple syntax that allows the user to define his own criterion based upon classical measures such an the degree, the number of edges inside a cluster, the number of edges crossing a cluster etc. as well as some basic operators such as add, substract, multiply, divide, sum. Each operator is associated to a vector of data either associated to vertices, clusters or the whole partition (the graph). As an example, the invariant INDEG_N represents the vector of inner degree associated to each vertex. Should a vertex of degree 4 have 3 neighbors in the same cluster, its value would be 3. In *Constellation*, INDEG_N represents a vector of size n indicating the inner degree of each vertex. In the same way, the invariant INCLS_CLS is a vector representing the number of edges between vertices of the same cluster. This vector could be considered as a vector of size k, having an entry for each cluster. The operators considered in *Constellation* involve 1 or 2 arguments. The result of the operator depends on its arguments. For example, multiplying a constant c by a vector v will give a vector of the same size as v. The operator SUM_CLS has as argument a vector of values corresponding to nodes and summarizes these values by cluster, thus providing a vector of size k. For the time being, *Constellation* involves relatively few operators and few invariants but this number is increasing with the usage to reach the users needs. Any line in the problem definition is associated to a vector resulting from its operator or invariant. Arguments are given by the number of the line where it is defined. The modularity definition in *Constellation* is defined as described on Figure 1. Line 1 is a vector of the number of edges within each cluster, that we shall call *in*. Line 2 is an operation which consists in multiplying each entry of *in* (the first argument is 1, which corresponds to *in*) by 2, the value associated to this operator. As a result, line 2 refers to a vector whose values are $2 * in$. Following the definitions, one finds the modularity associated to the current partition defined on line 11. To indicate to *Constellation* that line 11 refers to the objective function, the radio button on line 11 and column *Obj* is checked. In a similar manner, the user may define up to 3 constraints comparing the right hand side to the left hand side of the constraint sign (= or <= or >=).



	Name	Arg1	Oper/Inv	Arg2	Obj	Constr1	Constr2	Constr3	Value
1	in	0	INCLS_CLS	0	○	○	○	○	0,00
2	2in	1	MULTBY	0	○	○	○	○	2,00
3	out	0	OUTCLS_CLS	0	○	○	○	○	0,00
4	2in+out	2	PLUS	3	○	○	○	○	0,00
5	m	0	NBARCS_G	0	○	○	○	○	0,00
6	2m	5	MULTBY	0	○	○	○	○	2,00
7	(2in+out)/2m	4	DIV	6	○	○	○	○	0,00
8	v2	7	MULT	7	○	○	○	○	0,00
9	in/m	1	DIV	5	○	○	○	○	0,00
10	v2-in/m	8	MINUS	9	○	○	○	○	0,00
11	modularity	10	SUM	0	●	○	○	○	0,00
12		0	UNUSED	0	○	○	○	○	0,00
13		0	UNUSED	0	○	○	○	○	0,00
14		0	UNUSED	0	○	○	○	○	0,00
15		0	UNUSED	0	○	○	○	○	0,00
16		0	UNUSED	0	○	○	○	○	0,00
17		0	UNUSED	0	○	○	○	○	0,00
18		0	UNUSED	0	○	○	○	○	0,00

☐ Fix nb cls [✓ Max] Min

Figure 1: Definition of a problem in *Constellation*

To handle constraints and avoid numerical instability, the quality of a solution is not only characterized by the objective function value (as it would be by classical Lagrangian type penalization techniques), but by a vector of values sorted by increasing magnitude which are lexicographically compared during the optimization. The first entry is the objective function value, then are 3 values dedicated to constraints chosen by the user and a last value is dedicated to the number of clusters if it is fixed. The graphical interface does not display the values of the constraints but displays the partition quality in red if it corresponds to a non realizable solution.

For convenience, some special operators are provided in *Constellation* to model constraints. For example, the operator <= used in $a <= b$ will return $b - a$ is $a > b$ and 0 otherwise. Should the user select that this value corresponds to constraint 1, this value will be minimized with a higher importance than the original objective function.

As soon as a criterion is entered in *Constellation*, it may be saved and loaded again or modified as needed.

4 Variable Neighborhood Search

The software is designed to handle a wide variety of graph clustering problems, therefore, no specific routine that is well known to work for any given criterion is implemented. Instead, it was decided that generic routines would be used. As it was found in some other generic optimization software such as AutoGraphiX (AGX) [1, 2], the variable neighborhood search (VNS) heuristic [4, 5] provides good results for a wide variety of problems with little need for parameter adjustments. For this reason, the optimization uses VNS in *Constellation*.

4.1 VNS overview

The optimization in *Constellation* is done by VNS which is well suited to handle a wide variety of problems with little tuning, compared to most other methods such as tabu search or genetic algorithms.

Let S be a solution of the communities detection problem and consider a transformation T, that changes S to an alternate solution. The set of solutions obtained from S by applying T is called $N_T(S)$, the neighborhood of S according to transformation T.

Searching the neighborhood $N_T(S)$ to find a better solution may recursively be used to define a local search LS. If S^* is better than any of the solutions in $N_T(S^*)$, we say that S^* is a local optima with regard to T.

Next, consider an extension of $N_T(S)$, $N_T^k(S)$ which is defined by applying k times the transformation T instead of once. N_T^k will most probably consist in a larger set of solutions than N_T. By changing the value of k, we get different neighborhoods and the number of corresponding solutions tends to grow with k. We say that N^k is nested in N^{k+1}. An exhaustive search in a very wide neighborhood is very time consuming and usually not recommended but picking at random a solution in $N^k(S)$ (applying a perturbation P^k to S) will provide a new starting solution from which LS could be applied. Depending on the value of k, the search will be concentrated in a small or large vicinity of S.

Using these key concepts, one could construct a first VNS algorithm, alternating local search and variable magnitude perturbations as described on Figure 2.

```
Initialization:
  • Select the neighborhood structure $N^k$ and a stopping condition.
  • Let $S$ be an initial (usually random) partition.
  • Let $S^*$ denote the best partition obtained to date.
Repeat until condition is met:
  • Set $k \leftarrow 1$;
  • Until $k = k_{max}$,do:
      (a) Generate a random partition $S' \in N^k(S)$;
      (b) Apply LS to $S'$
      Denote $S''$ the obtained local optimum $S'' = LS(S')$;
      (c) If $S''$ is better than $S^*$,
              Let $S^* \leftarrow S''$ and
              $k \leftarrow 1$
          otherwise,
              set $k \leftarrow k + 1$.
  done
```

Figure 2: Rules of Variable Neighborhood Search.

4.2 Local Search

Local search is achieved by variable neighborhood descent. The user may choose the transformations involved in the search from a list. If more than one is selected, all the transformations are used by the system one after the other until none of them succeed in improving the quality of the solution.

After some experiments, we identified that few of the transformations are really efficient and we will concentrate on them here. For clarity reasons, we will refer to these local search by the name of the transformation involved *i.e., Merge, Split* and *Redispatch*. In all the local searches used by *Constellation*, the *first improvement* strategy is used, which means that as soon as a transformation is found interesting, it is applied even if another transformation would be more profitable. Indeed, it is possible that the global improvement after a step of the search is not as good as it would be with the *best improvement* which only applies the most interesting transformation after testing them all, but *first improvement* is faster to apply, so that, overall, it seems to be a good choice.

1. **Merge**
   In the *Merge* local search, all pairs of clusters are considered one after the other. Should merging two clusters improve the solution, this transformation is applied.
2. **Split**
   The *Split* local search could be considered as the reverse of *Merge*. Each cluster is considered for splitting and as soon as a split is found improving, it is applied. From a technical point of view, there are many ways to split a cluster. Some of those are not efficient and others are very time consuming. As the potential splits applied in the optimization process a fast yet efficient implementation was chosen. If we denote by c_i the cluster to split, two distinct non pending vertices (v_1 and v_2) belonging to the c_i are first selected and v_2 is assigned to an empty cluster c_j. Then, each vertex v of c_i is assigned to cluster c_i or c_j if it is closer to v_1 or v_2 respectively.
3. **Redispatch**
   The *Redispatch* transformation is very close to *Split*, the only difference being that instead of considering a cluster c_i, two clusters c_i and c_j are considered. For every pair of clusters (c_i, c_j), we first randomly choose two non pending vertices v_i and v_j belonging to $c_i \cup c_j$. Then, each observation from $c_i \cup c_j$ is reassigned to c_i or c_j depending if it is closer to v_i or v_j.

4.3 Perturbations

For the time being, 6 perturbation schemes are implemented in *Constellation* and the choice of the perturbation to use is left to the user. The selected perturbation is then successively applied k times to obtained a partition in $N^k(S)$.

- *Redispatch* Two clusters are randomly selected and the same *Redispatch* procedure as the one defined in the local search is applied.
- *Split* One cluster is randomly selected cluster and split it in two in the way described for the *Split* procedure in local search.
- *Move Star* One vertex is randomly chosen and moved altogether with all its neighbors to another cluster also randomly chosen.
- *Multistart* This perturbation is not sensitive to the value k and consists in assigning every vertex a random cluster to simulate the multistart procedure within the vns framework.

4.4 Multi threads support

The optimization algorithm was developed in order to take advantage of multithreading. The easiest way to use VNS in a multithreading scheme is by making a copy of the whole problem under study for each thread and only sharing the best known solution. This approach limits the access to the shared memory and is more efficient from this point of view. In *Constellation*, this procedure is implemented and each thread uses its own sets of parameters, so that different threads may use different local searches and different values for k_{max}

for example. Unfortunately, it was found that having different threads working on the same current solution with various local searches may provide interesting results. For this reason, the concept of **collaborative** threads was also implemented in *Constellation*. From a technical point of view, each thread is linked to a MASTER thread with which it shares the current solution. The master thread is then linked to itself and the system does not allow a thread to be linked to another that is not a master (that is not linked to itself). For example, threads 2 and 3 may be linked to thread 1 (the master) while threads 5 and 6 are linked to thread 4 (another master). In this case, threads 1, 2 and 3 are collaborating and threads 4, 5 and 6 also. In this example, two sets of threads are to be considered: $(1, 2, 3)$ and $(4, 5, 6)$ these two sets are not collaborating and may explore different spaces while threads from a same set are improving the same solution. The masters are the only threads that apply perturbations when reaching a local optima (the others will only search for the master's solution).

Collaborative threads improve the efficiency of the local search but are demanding because they share some information that is often read or written. It is recommended that small groups of threads collaborates while it is not a problem if there are a lot of threads that only share the best known solution. Indeed, even if threads are collaborating, they only share the current solution at local optima in order to avoid the excessive overhead time due to the waiting for the access to the memory. When a thread reaches a local optima, it searches for the current solution from the master. Should this solution be better, it is loaded and used as starting point for a local search. In the case the local solution is better than that from the master this last solution is updated for the use of the other threads.

5 Application examples

An example of problem study with *Constellation* is presented. The *netscience_main* dataset which gives a network of collaborations in research is considered as it represents an example of reasonable size for which the graphical representation is interesting and easy to visualize. *Constellation* allows a first drawing of the network in such a way that its structure is easy to understand, as shown on Figure 3. Note that the node size could be modified to provide information on the degree of the corresponding vertex.

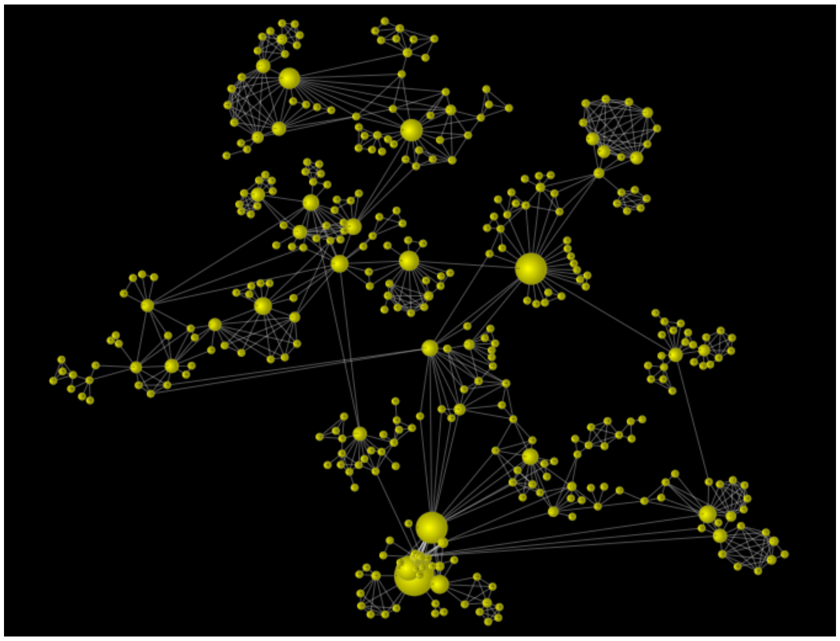


Figure 3: The netscience graph representation

5.1 Maximizing modularity

As an illustration of the use of *Constellation*, for finding communities in a graph, we consider an example of clustering criterion that is most likely the most used, the *modularity*. This criterion was initially proposed by Girvan and Newman in 2002 [3] as a stopping rule for a hierarchical divisive algorithm and was later proposed as an independent criterion [6].

Modularity aims at finding a partition of V which maximizes the sum, over all clusters (or modules), of the number of inner edges minus the expected number of such edges assuming that they are drawn at random with the same distribution of degrees as in G. *Modularity* is defined as:

$$Q = \sum_{C \in \mathcal{C}} \left[a_C - e_C \right],$$

where a_C is the fraction of all edges that lie within module C and e_C is the expected value of the same quantity in a graph in which the vertices have the same expected degrees but edges are placed at random. A maximum value of Q near to 0 indicates that the network considered is close to a random one (barring fluctuations), while a maximum value of Q near to 1 indicates strong community structure. Observe that maximizing modularity gives an optimal partition together with the optimal number of modules.

Let the vertex function weight function be defined as:

$$\omega(v) = \begin{cases} \displaystyle\sum_{\{u,v\} \in E} \omega(\{u,v\}) & \text{if } \{v,v\} \notin E \\ \displaystyle\sum_{\{u,v\} \in E, u \neq v} \omega(\{u,v\}) + 2\omega(\{v,v\}) & \text{if } \{v,v\} \in E \end{cases}$$

Let $\mathcal{C}$ be a partition of V, the sum over modules of their modularities can be written as

$$Q = \frac{\displaystyle\sum_{C \in \mathcal{C}} \sum_{\{u,v\} \in E_C} \omega(\{u,v\})}{\displaystyle\sum_{e \in E} \omega(e)} - \frac{\displaystyle\sum_{C \in \mathcal{C}} \left(\sum_{v \in V_C} \omega(v) \right)^2}{4 \left(\displaystyle\sum_{e \in E} \omega(e) \right)^2}. \tag{1}$$

The modularity is interesting as a starting point because the number of clusters does not need to be fixed.

The Figure 4 represents the solution obtained by *Constellation* for the modularity maximization criterion. On this solution, the value of the modularity is 0.848587 and has 19 clusters represented by different colors. Note that for technical reasons, the colors of different clusters may look similar, but it is easy to recognize these cases by looking at the underlying structure.

5.2 Maximizing modularity with Radicchi constraints

To improve the quality of the clustering obtained, some constraints could be added reinforcing the concept of clusters. Radicchi et al. [7] proposed some condition that a cluster should respect to be considered as a community in the strong sense. This condition is that each vertex must have more neighbors in its community than neighbors outside its community. When this condition is added to the problem definition, *Constellation* finds a different solution which only has 17 clusters and the value of the modularity was reduced to 0.812573. We notice some large clusters that seem to be the junction of clusters that are joined by one single common vertex.

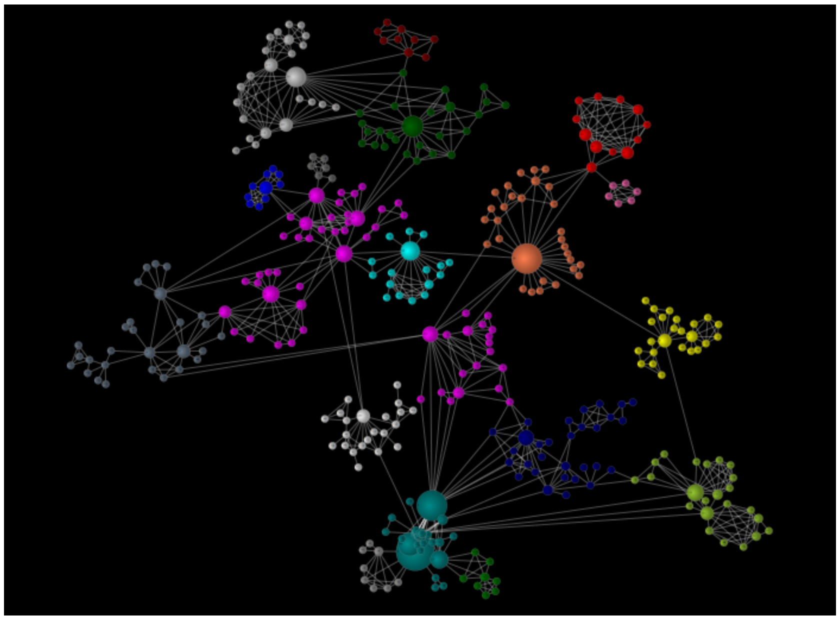


Figure 4: The netscience graph solution found by *Constellation* while maximizing modularity.

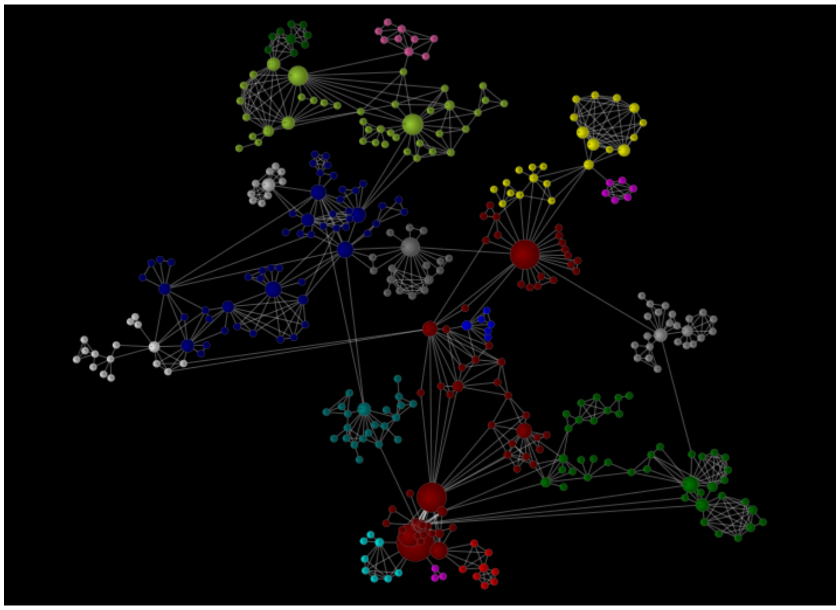


Figure 5: The netscience graph solution found by *Constellation* while maximizing modularity with strong Radicchi conditions.

References

[1] M. Aouchiche, J.-M. Bonnefoy, A. Fidahoussen, G. Caporossi, P. Hansen, J. Lacheré, and A. Monhait. Variable neighborhood search for extremal graphs. 14. The AutoGraphiX 2 system. In L. Liberti and N. Maculan, editors, *Global Optimization. From Theory to Implementation.* Springer Science, New-York, 2006.

[2] G. Caporossi and P. Hansen. Variable neighborhood search for extremal graphs. 1. The AutoGraphiX system. *Discrete Math.*, 212:29–44, 2000.

[3] M. Girvan and M. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. U.S.A.*, 99:7821–7826, 2002.

[4] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European J. Oper. Res.*, 130:449–467, 2001.

[5] N. Mladenović and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24:1097–1100, 1997.

[6] M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026133, 2004.

[7] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto and D. Parisi. Defining and identifying communities in networks. *PNAS*, 101:2658–2663, 2004.