Clustering on networks by modularity maximization

Sonia Cafieri

ENAC – Ecole Nationale de l'Aviation Civile
Toulouse, France

thanks to:

Pierre Hansen, Sylvain Perron, Gilles Caporossi (GERAD, HEC Montréal, Canada)

Leo Liberti (École Polytechnique, France)

INRA - Toulouse, March 2012

Outline

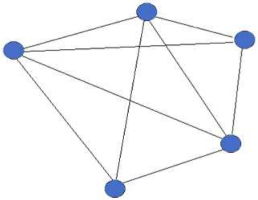
1. Community identification in Networks
2. Modularity maximization
   - Definition & State of the art
3. Exact algorithms for modularity maximization
   - Modularity maximization as clique partitioning
   - Modularity maximization by mixed 0-1 quadratic programming
4. Locally optimal hierarchical divisive heuristic
   - Hierarchical schemes
   - An exact algorithm for bipartition & a new divisive heuristic
5. Refinement of heuristic results
   - Improving heuristic by merging+splitting
6. Modularity maximization on trees
   - Dynamical-Programming based algorithm
7. Other research directions

Networks often used to represent complex systems

A network, or graph: $G = (V, E)$

V = *Vertices*, associated with the entities of the system under study (people, companies, towns, natural species, ...).
represented by points

E = *Edges*, express that a relation defined on all pairs of vertices holds or not for each such pair
represented by lines joining vertices



Network Clustering

Networks often used to represent complex systems

- social networks
- telecommunication networks
- transportation networks
- ...

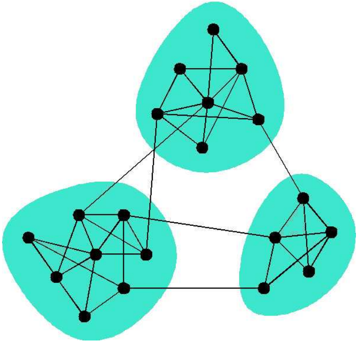
Communities

Automatic analysis of complex systems represented as networks

$\Downarrow$

$\boxed{\text{identification of communities}}$

community = a subset of vertices such that there are
more edges within the community than edges joining it to the outside



Partitions

A community corresponds to a subgraph $G_S = (S, E_S)$ of a graph $G = (V, E)$:
a graph with vertex set $S \subseteq V$, edge set E_S equal to all edges with both vertices in S.

One aims at finding a partition of V into subgraphs induced by nonempty subsets

$$V_1, V_2, \ldots, V_M$$

such that

$$V_k \cap V_l = \emptyset \quad \forall k \in 1, 2, \ldots M$$

$$V_1 \cup V_2 \cup \ldots \cup V_M = V$$

.

How to evaluate a partition?

$\Downarrow$

we need a clustering criterion / definition of community

Clustering criteria

- Minimum cut:

$$\min_{C_1,\ldots,C_k} \sum_{s=1}^{k} links(C_s, V \backslash C_s)$$

- Ratio cut (Cheng and Wei, 1991):

$$\min_{C_1,\ldots,C_k} \sum_{s=1}^{k} \frac{links(C_s, V \backslash C_s)}{|C_s|}$$

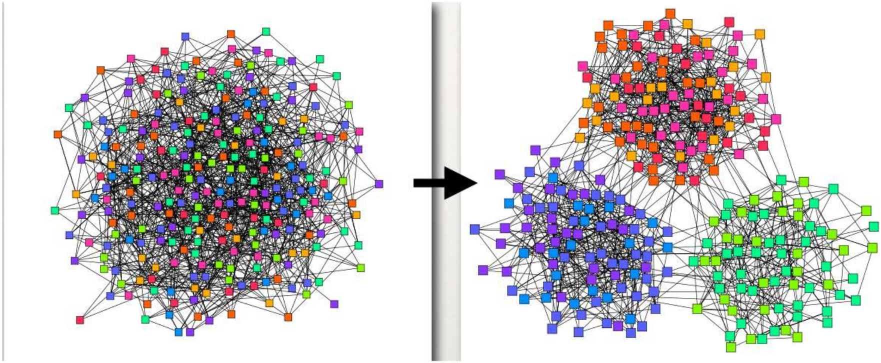
- Normalized cut (Shi and Malik, 2000):

$$\min_{C_1,\ldots,C_k} \sum_{s=1}^{k} \frac{links(C_s, V \backslash C_s)}{degree(C_s)}$$

- Min-max cut (Ding et al., 2001):

$$\min_{C_1,\ldots,C_k} \sum_{s=1}^{k} \frac{links(C_s, V \backslash C_s)}{links(C_s, C_s)}$$

Clustering criteria



Outline

Modularity

Newman and Girvan, 2004:

compare the fraction of edges falling within communities
to the expected fraction of such edges

Modularity:

$$Q = \sum_{s} [a_s - e_s]$$

a_s = fraction of all edges that lie within module s

e_s = expected value of the same quantity in a graph in which the vertices have the same degrees but edges are placed at random.

Modularity

Newman and Girvan, 2004:

compare the fraction of edges falling within communities
to the expected fraction of such edges

Modularity:

$$Q = \sum_s [a_s - e_s]$$

a_s = fraction of all edges that lie within module s

e_s = expected value of the same quantity in a graph in which the vertices have the same degrees but edges are placed at random.

- $Q \approx 0$: the network is equivalent to a random network (barring fluctuations);
- $Q \approx 1$: the network has a strong community structure;
- in practice, the maximum modularity Q is often between 0.3 and 0.7.

Maximizing modularity gives an optimal partition with the optimal number of clusters

Modularity maximization: methods

- Exact algorithms for modularity maximization
  - proposed only in a few papers
  - can only solve small instances (with about a hundred entities) in reasonable time
  - provide an optimal solution together with the proof of its optimality
- Heuristics for modularity maximization
  - widely used
  - can solve approximately very large instances with up to hundred or thousand entities
  - do not have either an a priori performance guarantee
    (finding always a solution with a value which is at least a given percentage of the optimal one),
    nor an a posteriori performance guarantee
    (that the obtained solution is at least a computable percentage of the optimal one)

Modularity maximization: methods

Heuristics based on:

Partitioning schemes
aim at finding the best partition into a given number of clusters

- simulated annealing,
- genetic search,
- multistep greedy,
- a variety of other approaches.

Hierarchical clustering
lead to a set of nested partitions

- agglomerative schemes
- divisive schemes

Modularity literature: successful approaches

- Clauset, Newman and Moore, 2004:
  agglomerative hierarchical greedy, for sparse networks has a very low complexity and is considerably faster than previously proposed methods.
- Newman, 2006:
  divisive hierarchical heuristic based on spectral graph theory,
  splitting is done according to the sign of the components of the first eigenvector of the modularity matrix.
- Noack and Rotta, 2009:
  heuristic based on a single-step coarsening with a multi-level refinement, competitive with other methods in the literature.
- Liu and Murata, 2010:
  heuristic based on label propagation, gives better results than previous heuristics

Criticism to modularity

- Resolution limit:

  in the presence of large clusters, some clusters smaller than a certain size can be undetectable $\Rightarrow$ modular structures like small cliques can be hidden in larger clusters.

- Degeneracy of Q:

  there can be a large number of partitions, even very different from each other, having high modularity values $\Rightarrow$ easy to find high-scoring partitions but difficult to identify the global optimum.

Our contribution on modularity maximization

- Exact algorithms:
  row generation, column generation
  $\Rightarrow$ raising the size of exact solved problems

- Heuristic:
  locally optimal hierarchical divisive heuristic

- Refinement of heuristic results

- Dynamical-Programming based algorithm for modularity maximization on trees

Outline

Modularity: another expression

Modularity as a sum of values over all edges of the complete graph K_n:

$$Q = \frac{1}{2m} \sum_{i,j \in V} \left(a_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j)$$

where:

- m = $|E|$
- k_i, k_j = degrees of vertices i and j
- a_{ij} = ij component of the adjacency matrix of G
- $\delta(c_i, c_j)$ = 1 if the communities to which i and j belong are the same, 0 otherwise (Kronecker symbol)
- $k_i k_j / 2m$ = expected number of edges between vertices i and j in a null model where edges are placed at random and the distribution of degrees remains the same.

Modularity maximization as clique partitioning

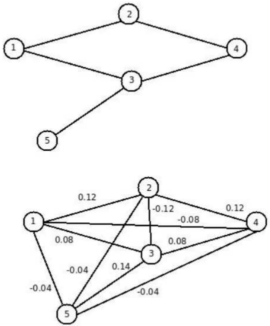
Introducing binary variables

$$\begin{cases} x_{ij} = 1 & \text{if vertices } i,j \text{ belong to the same community} \\ \phantom{x_{ij}} = 0 & \text{otherwise} \end{cases}$$

and

$$w_{ij} = \frac{1}{m}\left(a_{ij} - \frac{k_i k_j}{2m}\right)$$

modularity maximization can be reformulated as a clique partitioning problem:



- m= 5
- d_1= 2
- d_2= 2
- d_3= 3
- d_4= 2
- d_5= 1
- a_{12}= 1, m = 5, d_1= 2, d_2= 2
- $\Rightarrow w_{12} = \frac{1}{5}\left(1 - \frac{2\times 2}{10}\right) = 0.12$

Modularity maximization as clique partitioning

modularity maximization can be reformulated as a clique partitioning problem:

K_n complete graph $\Rightarrow$ it is a clique and any of its induced subgraphs are cliques.

Partitioning G is thus equivalent to partitioning K_n into cliques.

The resulting partition is an equivalence relation:

- reflexivity: each entity is in the same module as itself: $\forall i \quad x_{ii} = 1$
- symmetry: if i is in the same module as j, j is in the same as i: $\forall i, j \quad x_{ij} = x_{ji}$
- transitivity: if i and j are in the same module and j and k are in the same module, then i and k must be in the same module

Modularity maximization as clique partitioning

$$\begin{cases} \max & \sum_{i<j\in V} w_{ij}x_{ij} - C & -C = -\sum_{i\in V}\frac{k_i k_i}{2m} \\ \text{s.t.} & x_{ij} + x_{jk} - x_{ik} \leq 1 & \forall 1 \leq i < j < k \leq n \\ & x_{ij} - x_{jk} + x_{ik} \leq 1 & \forall 1 \leq i < j < k \leq n \\ & -x_{ij} + x_{jk} + x_{ik} \leq 1 & \forall 1 \leq i < j < k \leq n \\ & x_{ij} \in \{0,1\} & \forall 1 \leq i < j \leq n \end{cases}$$

(Grötschel and Wakabayashi, 1990)

Linear 0-1 program

$$\frac{n(n-1)}{2} = O(n^2) \text{ variables}$$

$$3\begin{pmatrix} n \\ 3 \end{pmatrix} = \frac{n(n-1)(n-2)}{2} = O(n^3) \text{ constraints}$$

Row generation

Typically used in combinatorial applications.

1. the linear continuous relaxation is first solved
2. if the solution of this relaxation is in integers, it is optimal (often the case for modularity maximization)
3. if the solution of the continuous relaxation is fractional, add valid constraints violated by the fractional solution: *cutting planes*
4. the number of constraints grows rapidly with n: they can be added by batches of unsatisfied ones.

Our solution of the Linear 0-1 program:

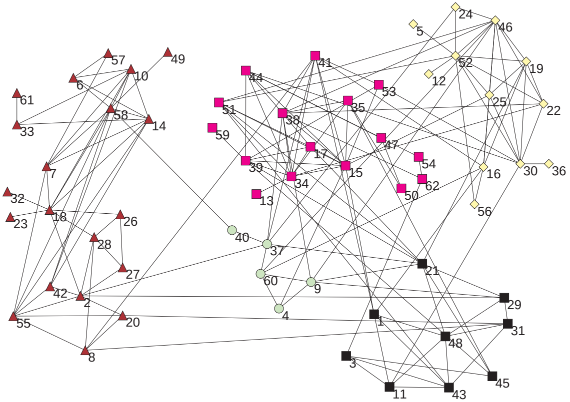
- `CPLEX`
- row generation approach implemented with the "lazy constraints" feature of `CPLEX`.

Example: a social network

Dolphins network:

bottlenose dolphins studied by Lusseau in Doubtful Sound, New Zealand.

Network with 62 vertices corresponding to the dolphins and 159 edges joining vertices associated with pairs of dolphins with frequent communications among them.



partition obtained for *dolphins* dataset.

Column generation

It is a powerful technique of linear programming which allows solving exactly linear programs with a number of variables (columns) exponential in the size of the input.

Basic steps:

1. select a small number of columns and solve the linear program using only these
2. find an unused column which, if included, would most improve the objective value (with favorable *reduced cost*) or determine that there is none
3. include the column in the linear program, re-solve it, and go to step 2.

The original problem is split into:

- Master problem:
  original problem with only a subset of variables being considered
- Subproblem:
  new problem created to identify a new variable

Column generation – clique partitioning

Modularity maximization: the columns correspond to all subsets of V (all nonempty modules).

$$
\begin{aligned}
a_{it} &= 1 \quad \text{if vertex } i \text{ belongs to module } t \\
&= 0 \quad \text{otherwise}
\end{aligned}
$$

Master problem:

$$
\begin{cases}
\max \sum_{t \in T} c_t z_t - C & \\
\sum_{t \in T} a_{it} z_t = 1 & \forall i = 1, \ldots, n \\
z_t \in \{0, 1\} & \forall t \in T
\end{cases}
$$

$$c_t = \sum_i \sum_{j>i} w_{ij} a_{it} a_{jt}$$

i.e., the value of the module indexed by t, $t = 1 \ldots 2^n - 1$.

- obj. func.: sum of modularities of all selected modules minus the constant corresponding to the diagonal terms
- 1st set of constrains: each entity must belong to one and only one module
- 2nd set of constraints: modules must be selected entirely or not at all.

Column generation – clique partitioning

Improving columns are added progressively to relaxation of the master problem.

Reduced cost associated with column t: $c_t - \sum_i \lambda_i a_{it}$.

To find a column with positive red. cost, we replace the coefficients a_{it} by variables y_i.

Auxiliary problem:

$$\max \sum_i \sum_{j>i} w_{ij} y_i y_j - \sum_i \lambda_i y_i .$$

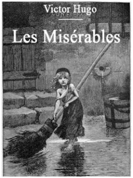
Quadratic program in 0-1 variables with a 100% dense matrix of coefficients

Solved using

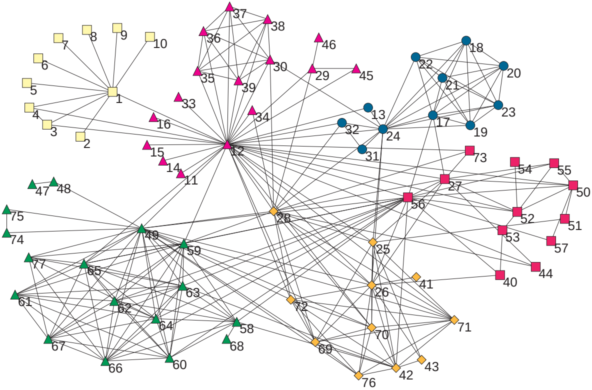
- a Variable Neighborhood Search heuristic
  (as long as a column with positive reduced cost can be found);
- as exact method, a simple branch and bound algorithm (Meyer 2000)
  (when this is no more the case).

Example: a social network

Victor Hugo's *Les Misérables* network:



describes the relationships between characters in Victor Hugo's masterpiece : 77 vertices associated to characters which interact and 257 edges associated with pairs of characters appearing jointly in at least one chapter.



Outline

MIQP formulation

Xu, Tsoka and Papageorgiou (2007):

$$Q = \sum_s [a_s - e_s] = \sum_s \left[\frac{m_s}{m} - \left(\frac{d_s}{2m} \right)^2 \right]$$

m_s = number of edges in module s
d_S = sum of degrees k_i of the vertices of module s.

Variables used to identify to which module each vertex and each edge belongs:

$$X_{rs} = \begin{cases} 1 & \text{if edge } r \text{ belongs to module } s \\ 0 & \text{otherwise} \end{cases} \qquad \forall r = 1, 2, \ldots m, \; s = 1, 2, \ldots M$$

$$Y_{is} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to module } s \\ 0 & \text{otherwise.} \end{cases} \qquad \forall i = 1, 2, \ldots n, \; s = 1, 2, \ldots M$$

$$m_s = \sum_r X_{rs} \quad \text{and} \quad d_S = \sum_i k_i Y_{is}$$

MIQP formulation

- Each vertex belongs to exactly one module:
$$\sum_s Y_{is} = 1 \quad \forall i = 1, 2, \dots n$$

- Any edge $r = \{v_i, v_j\}$ with end vertices indiced by i and j can only belong to module s if both of its end vertices belong also to that module:
$$X_{rs} \leq Y_{is} \quad \forall r = \{v_i, v_j\} \in E$$
$$X_{rs} \leq Y_{js} \quad \forall r = \{v_i, v_j\} \in E$$

- The number of modules is *a priori* unknown.
Variables $u_s = 1$ if module s is nonempty and 0 otherwise. Then constraints $u_s \leq u_{s-1}$ express that module s can be nonempty only is module $s-1$ is so. Consequently:
$$\sum_r X_{rs} \geq u_s \quad \text{and} \quad \sum_r X_{rs} \leq (n - s + 1)u_s$$
($n - s + 1$ due to the fact that each of the modules $1, 2, \dots s - 1$ must be nonempty).

- Alternative equivalent solutions can be obtained by simply re-indexing clusters
$\Rightarrow$ symmetry-breaking constraints.

MIQP formulation

$$\Downarrow$$

Mixed-Integer Quadratic Program

with a convex continuous relaxation

Solved using `CPLEX`.

Instances up to 104 vertices are solved.

MIQP – column generation

Master problem: the same as the first CG approach

Auxiliary problem: mixed 0-1 quadratic program, approach similar to that one of Xu et al.:

$$\begin{cases} \max & \displaystyle\sum_r \frac{x_r}{m} - \left(\frac{d}{2m}\right)^2 - \sum_i \lambda_i y_i. \\ s.t. & \\ d = & \displaystyle\sum_i k_i y_i \\ x_r \leq & y_i \quad \forall r = \{v_i, v_j\} \in E \\ x_r \leq & y_j \quad \forall r = \{v_i, v_j\} \in E \end{cases}$$

$$\begin{cases} x_r = 1 & \text{if edge } r \text{ belongs to the module which maximizes the obj.function} \\ = 0 & \text{otherwise} \end{cases}$$

$$\begin{cases} y_i = 1 & \text{if vertex } i \text{ belongs to the module which maximizes the obj.function} \\ = 0 & \text{otherwise} \end{cases}$$

$$\Downarrow$$

Mixed-Integer Quadratic Program

$n + m$ binary variables + 1 continuous variable
$2m + 1$ linear constraints
a single nonlinear term which is concave, in the objective function.

Solved using:

- a Variable Neighborhood Search heuristic
  (as long as a column with positive reduced cost can be found);
- `CPLEX` (when this is no more the case).

Computational results

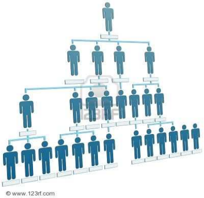
- clique partitioning row generation: **CPRG**
- clique partionning column generation: **CPCG**
- 0-1 mixed integer programming: **MIQP**
- 0-1 mixed integer column generation: **MICG**

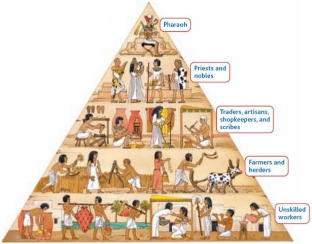
Comparison of CPU time

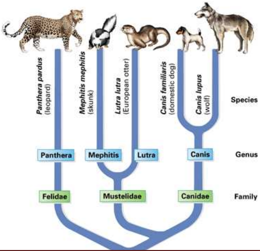
dataset	n	m	opt	M	CPRG	CPCG	MIQP	MICG
karate	34	78	0.4198	4	**0.02**	0.62	1.03	1.97
dolphin	62	159	0.5285	5	4.85	**2.96**	197.89	4.13
misérables	77	254	0.5600	6	**1.49**	1.70	55.58	1.63
p53	104	226	0.5351	7	601.69	**2.80**	1844.31	3.61
polbooks	105	441	0.5272	5	647.22	139.17	-	**35.78**
football	115	613	0.6046	10	**193.50**	-	-	204.50
s838	512	819	0.8194	12	-	-	-	**7655.56**

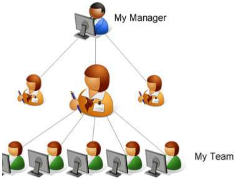
Outline

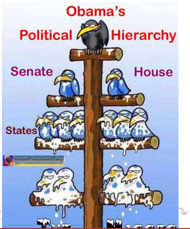
Hierarchies











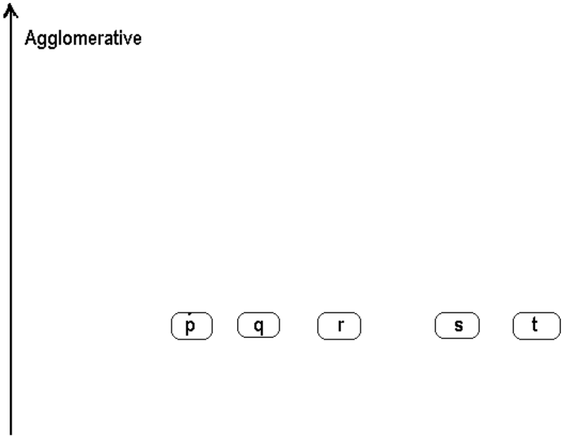
Hierarchical heuristics

Hierarchical heuristics are in principle devised for
finding a hierarchy of partitions implicit in the given network
when it corresponds to some situation where hierarchy is observed or postulated.
They aim at finding a set of nested partitions.

- Agglomerative heuristics
- Divisive heuristics

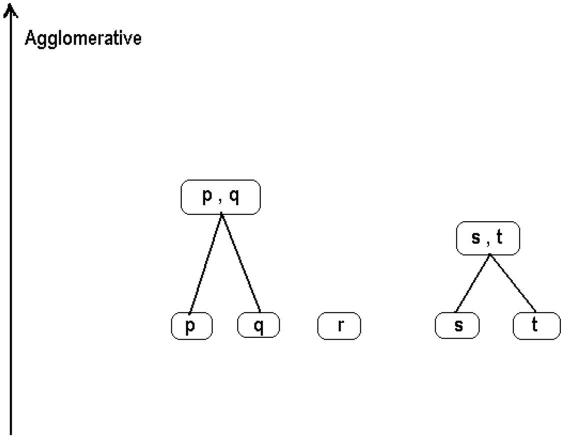
Agglomerative and Divisive heuristics

- Agglomerative heuristics
  - proceed from an initial partition with n communities each containing 1 entity
  - iteratively merge the pair of entities for which merging increases most the objective function (e.g., modularity)



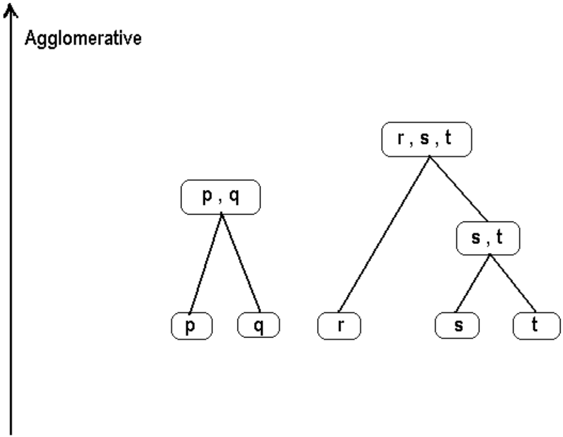
Agglomerative and Divisive heuristics

- Agglomerative heuristics
  - proceed from an initial partition with *n* communities each containing 1 entity
  - iteratively merge the pair of entities for which merging increases most the objective function (e.g., modularity)



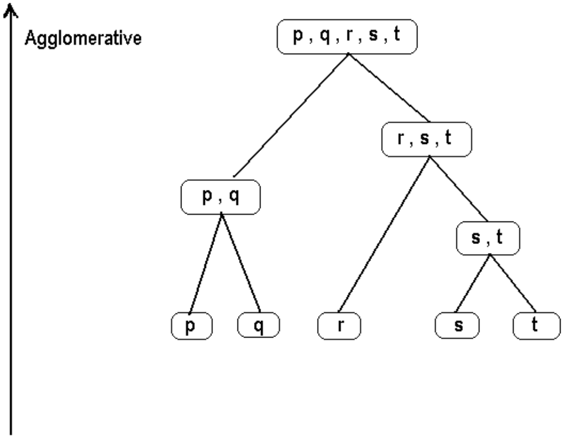
Agglomerative and Divisive heuristics

- Agglomerative heuristics
  - proceed from an initial partition with n communities each containing 1 entity
  - iteratively merge the pair of entities for which merging increases most the objective function (e.g., modularity)



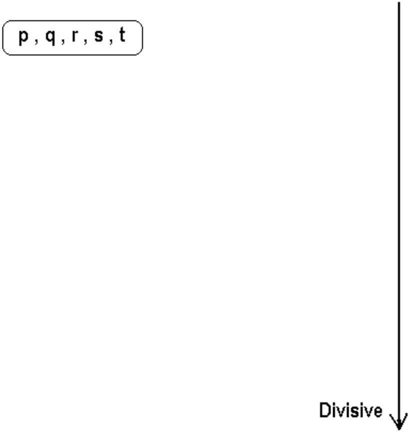
Agglomerative and Divisive heuristics

- Agglomerative heuristics
  - proceed from an initial partition with *n* communities each containing 1 entity
  - iteratively merge the pair of entities for which merging increases most the objective function (e.g., modularity)



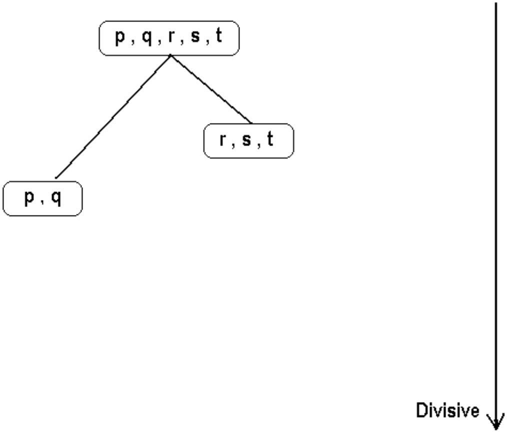
Agglomerative and Divisive heuristics

- Divisive heuristics
  - proceed from an initial partition containing all entities
  - iteratively divide a community into two in such a way to increase most the objective function (or the decrease in the objective value is the smallest possible).



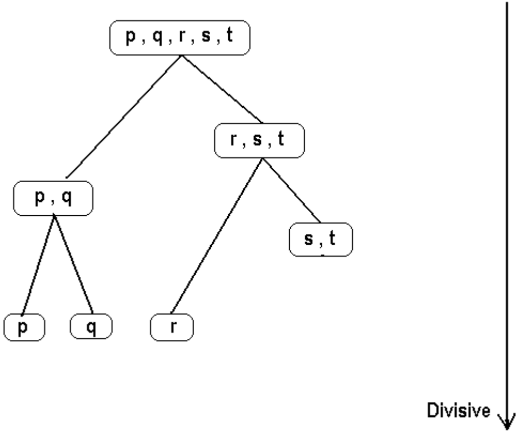
Agglomerative and Divisive heuristics

- Divisive heuristics
  - proceed from an initial partition containing all entities
  - iteratively divide a community into two in such a way to increase most the objective function (or the decrease in the objective value is the smallest possible).



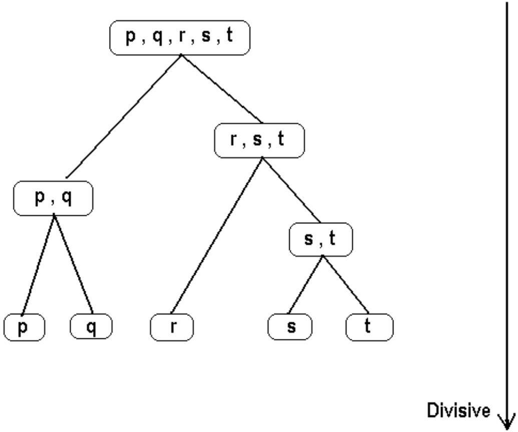
Agglomerative and Divisive heuristics

- Divisive heuristics
  - proceed from an initial partition containing all entities
  - iteratively divide a community into two in such a way to increase most the objective function (or the decrease in the objective value is the smallest possible).



Agglomerative and Divisive heuristics

- Divisive heuristics
  - proceed from an initial partition containing all entities
  - iteratively divide a community into two in such a way to increase most the objective function (or the decrease in the objective value is the smallest possible).



Outline

An exact algorithm for bipartition

$$Q = \sum_s [a_s - e_s] = \sum_s \left[\frac{m_s}{m} - \left(\frac{d_s}{2m} \right)^2 \right]$$

m_s = number of edges in community s
d_S = sum of degrees k_i of the vertices of community s

we aim to find a bipartition $\rightarrow$ $s \in \{1, 2\}$

Variables used to identify to which module each vertex and each edge belongs:

$$X_{rs} = \begin{cases} 1 & \text{if edge } r \text{ belongs to community } s \\ 0 & \text{otherwise} \end{cases} \qquad \forall r = 1, 2, \ldots m,\ s = 1, 2$$

$$Y_{i1} = \begin{cases} 1 & \text{if vertex } i \text{ belongs to community } 1 \\ 0 & \text{otherwise, i.e. if vertex } i \text{ belongs to community } 2 \end{cases} \qquad \forall i = 1, 2, \ldots n.$$

An exact algorithm for bipartition

We express d_2 as a function of d_1:

$$d_2 = d_t - d_1$$

(d_t = sum of degrees in the community to be bipartitioned).

$\Rightarrow$ Modularity:

$$\begin{aligned} Q &= \frac{m_1 + m_2}{m} - \frac{d_1^2}{4m^2} - \frac{d_2^2}{4m^2} = \\ &= \frac{m_1 + m_2}{m} - \frac{d_1^2}{4m^2} - \frac{d_t^2 + d_1^2 - 2d_t d_1}{4m^2} = \\ &= \frac{m_1 + m_2}{m} - \frac{d_1^2}{4m^2} - \frac{d_t^2}{4m^2} + \frac{d_t d_1}{2m^2}. \end{aligned}$$

An exact algorithm for bipartition

We impose that any edge $r = \{v_i, v_j\}$ with end vertices indiced by i and j can only belong to community s if both of its end vertices belong also to that community:

$$\begin{array}{rcll} X_{r1} & \leq & Y_{i1} & \forall r = \{v_i, v_j\} \in E \\ X_{r1} & \leq & Y_{j1} & \forall r = \{v_i, v_j\} \in E \end{array}$$

and

$$\begin{array}{ll} X_{r2} \leq 1 - Y_{i1} & \forall r = \{v_i, v_j\} \in E \\ X_{r2} \leq 1 - Y_{j1} & \forall r = \{v_i, v_j\} \in E \end{array}$$

Furthermore:

$$m_s = \sum_r X_{rs} \quad \forall s \in \{1, 2\}$$

$$d_1 = \sum_{i \in V_1} k_i Y_{i1}.$$

An exact algorithm for bipartition

$$\Downarrow$$

Mixed-Integer Quadratic Program

with a single non linear but concave term, in the objective function,

which is to be maximized. Hence, its continuous relaxation is easy to solve.

Solved using `CPLEX`.

A new locally optimal divisive heuristic : *divisive CHL*

- Divisive scheme
- splitting step performed using the above exact algorithm for bipartition

$$\Downarrow$$

the proposed heuristic is locally optimal

(but not globally optimal)

Results: comparison with *CNM*

dataset	n	m	agglomerative CNM			divisive CHL			exact	
			M	Q	error(%)	M	Q	error(%)	M	Q
`karate`	34	78	3	0.38067	9.31895	4	0.41880	0.23583	4	0.41979
`dolphin`	62	159	4	0.49549	6.24953	4	0.52646	0.38977	5	0.52852
`les miserables`	77	254	5	0.50060	10.6087	8	0.54676	2.36603	6	0.56001
`A00_main`	83	135	7	0.52394	1.31098	7	0.52806	0.53494	9	0.53090
`p53 protein`	104	226	8	0.52052	2.73018	7	0.52843	1.25203	7	0.53513
`political_books`	105	441	4	0.50197	4.79288	4	0.52629	0.18018	5	0.52724
`football`	115	613	7	0.57728	4.51395	10	0.60091	0.60539	10	0.60457
`A01_main`	249	635	12	0.59908	5.34366	15	0.62877	0.65255	14	0.63290
`usair97`	332	2126	7	0.32039	12.9848	8	0.35959	2.33840	6	0.36820
`netscience_main`	379	914	19	0.83829	1.21494	20	0.84702	0.18619	19	0.84860
`s838`	512	819	12	0.80556	1.68904	15	0.81663	0.33805	12	0.81940
`power`	4941	6594	39	0.93402	–	40	0.93937	–	-	–
average			8	0.55125	5.52342	9.3	0.57525	0.82540	8.8	0.57957

Results: comparison with *divisive spectral + KL*

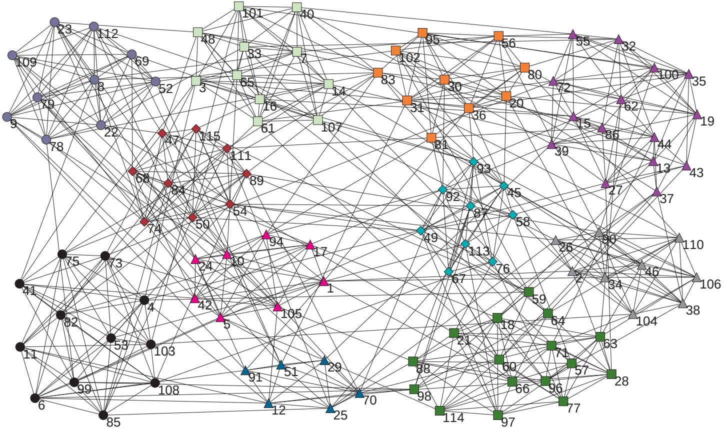
dataset	*n*	*m*	*divisive spectral + KL*				*divisive CHL*		
			M	*Q*	*err_dv*(%)	*error*(%)	*M*	*Q*	*error*(%)
`karate`	34	78	4	0.419	0	0.236	4	0.41880	0.23583
`dolphin`	62	159	5	0.508	3.415	3.792	4	0.52646	0.38977
`les_miserables`	77	254	7	0.538	1.533	3.862	8	0.54676	2.36603
`A00_main`	83	135	7	0.527	0.199	0.733	7	0.52806	0.53494
`p53 protein`	104	226	6	0.518	1.930	3.158	7	0.52843	1.25203
`political_books`	105	441	4	0.527	-0.081	0.099	4	0.52629	0.18018
`football`	115	613	8	0.579	3.638	4.221	10	0.60091	0.60539
`A01_main`	249	635	16	0.594	5.463	6.080	15	0.62877	0.65255
`usair97`	332	2126	7	0.358	0.501	2.827	8	0.35959	2.33840
`netscience_main`	379	914	23	0.820	3.191	3.371	20	0.84702	0.18619
`s838`	512	819	13	0.779	4.587	4.910	15	0.81663	0.33805
`power`	4941	6594	8	0.791	–		40	0.93937	–
average			9.09	0.56073	2.21592	3.02627	9.3	0.57525	0.82540

Example: a social network

Football game network

it describes the schedule of games between American college football teams in the Fall 2000.

$n = 115$ vertices, $m = 613$ edges.



Outline

Improving a partition: basic idea

Given a partition found by a heuristic:

- act on the reduced networks represented by the communities found
- merge and split some communities if this is worthwhile in terms of increase of modularity
- In particular:

  apply an exact algorithm for bipartitioning to split a community

$\Rightarrow$ Impact of exact algorithms on heuristic schemes

Merging + splitting communities

Post-processing to available heuristics for modularity maximization

$\Rightarrow$ the initial partition is the solution provided by the considered heuristic.

- First: **split** communities
  - split each CL_i of the original partition into 2 sub-communities CL_1, CL_2 by applying the exact algorithm for bipartition
  - **if** $Q(CL_1) + Q(CL_2) > Q(CL_i)$ **then**

    replace CL_i by the 2 new communities CL_1, CL_2

    else

    keep the original community CL_i.

Merging + splitting communities

- Second: **merge** pairs of communities
  - for each pair (CL_j, CL_k), merge CL_j and CL_k into CL_m
  - **if** $Q(CL_m) > Q(CL_j) + Q(CL_k)$ **then**

    replace CL_j, CL_k with $CL_m = CL_j \cup CL_k$

    else

    keep the original communities.

Merging + splitting communities

- Third: **merge + split** communities
  - for each pair (CL_j, CL_k), merge CL_j and CL_k into CL_m
  - **if** $Q(CL_m) > Q(CL_j) + Q(CL_k)$ **then**

    replace CL_j, CL_k with $CL_m = CL_j \cup CL_k$

    else

    split CL_m into CL_{m1}, CL_{m2}

    if $Q(CL_{m1}) + Q(CL_{m2}) > Q(CL_m)$ **then**

    replace CL_m with CL_{m1}, CL_{m2}

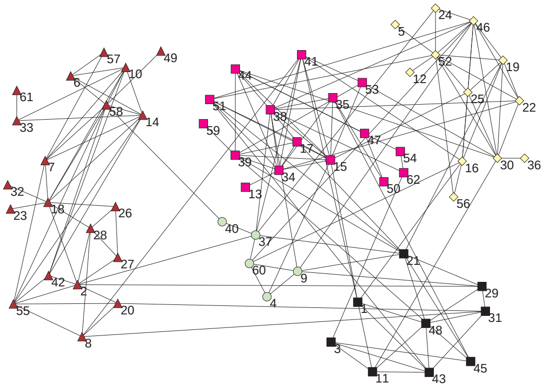
    else

    keep CL_m

Dolphins network:

bottlenose dolphins studied by Lusseau in Doubtful Sound, New Zealand.

Network with 62 vertices corresponding to the dolphins and 159 edges joining vertices associated with pairs of dolphins with frequent communications among them.



original: $Q = 0.52377$

split : $Q = 0.52773$

merge+split : $Q = 0.52852$

Results: comparison of modularities

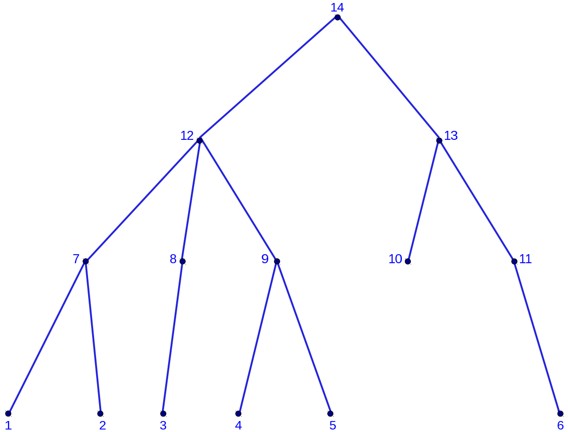
dataset	n	m	Q_{CNM}	Q'	Q_{NR}	Q''	Q_{opt}
`dolphin`	62	159	0.49549	0.51958	0.52377	**0.52852**	**0.52852**
`les miserables`	77	254	0.50060	0.54039	0.56001	**0.56001**	**0.56001**
`p53 protein`	104	226	0.52052	0.52621	0.53216	0.53502	0.53513
`political books`	105	441	0.50197	**0.52724**	0.52694	**0.52724**	**0.52724**
`adjnoun`	112	425	0.29349	0.29446	0.30729	0.30848	–
`football`	115	613	0.57728	0.58685	0.60028	**0.60457**	**0.60457**
`usair97`	332	2126	0.32039	0.36161	0.36577	0.36605	0.3682
`s838`	512	819	0.80556	0.80914	0.81624	0.81656	0.8194
`email`	1133	5452	0.51169	0.53808	0.57740	0.57773	–
`power`	4941	6594	0.93402	0.93612	0.93854	0.93870	–
`erdos02`	6927	11850	0.78092	0.78095	0.71552	0.71570	–

we transform some partitions into optimal ones !

Outline

Algorithm: labeling nodes and edges

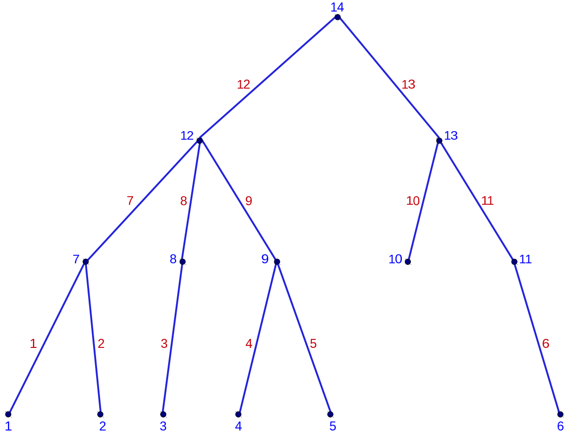
The algorithm first proceeds to labeling of its vertices and edges:



a center of the tree is found and vertices are given a level equal to the distance to the center and labeled accordingly beginning at the lowest level

Algorithm: labeling nodes and edges

The algorithm first proceeds to labeling of its vertices and edges:



edges are labeled with the same label as their lower vertex

Algorithm: triplets

Lists of triplets are associated with edges

they characterize the situation relative to the edges they are associated with and to the subtree rooted at their lower vertices.

A triplet (m_s, d_s, q_s):

- m_s = number of edges in the connected component containing the upper vertex v of the edge with which the triplet is associated;
- d_s = sum of degrees of this subtree;
- q_s = sum of modularities of the clusters within the subtree and not containing v.

Algorithm: triplets update

Two operations to update the set of triplets:

extension and **merging**

- **Extension**:
  considers the effect of adding or not an edge (u, v) to the subtree rooted at v
- **Merging**:
  considers the effect of combining two at a time, in increasing order of labels, two subtrees rooted at the same vertex v.

Extension:
considers the effect of adding or not an edge (u, v) to the subtree rooted at v

2 cases:

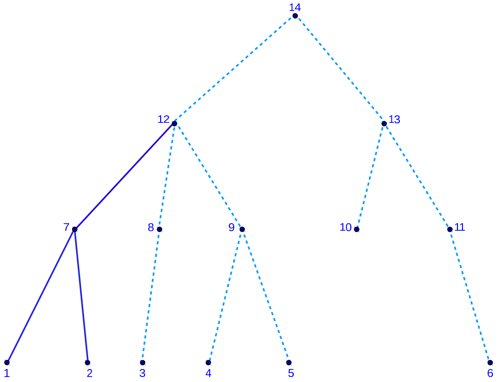
- the edge is cut
- the edge is not cut

Algorithm: extension

Extension:
considers the effect of adding or not an edge (u, v) to the subtree rooted at v

2 cases:

- the edge is cut
- the edge is not cut



Example: edge $(7, 12)$

⋆ if edge is cut $\rightarrow$ cluster [1,2,7]
with
edges=2, sum_degrees=5,
$Q_{[1,2,7]} = \frac{m_s}{m} - \left(\frac{d_s}{2m}\right)^2 = \frac{1}{9}$

⋆ if edge is not cut $\rightarrow$ contribution of the subtree $\{1, 2, 7, 12\}$ to the cluster containing the root 12: edges$\geq$ 3, sum_degrees= 5+degree of the root

Algorithm: triplets update

Extension:

- the edge is cut:
  the triplet (m_s, d_s, q_s) is transformed into

$$\left(0,\ d_{newroot},\ q_s + \left(\frac{m_s}{m} - \left(\frac{d_s}{2m}\right)^2\right)\right)$$

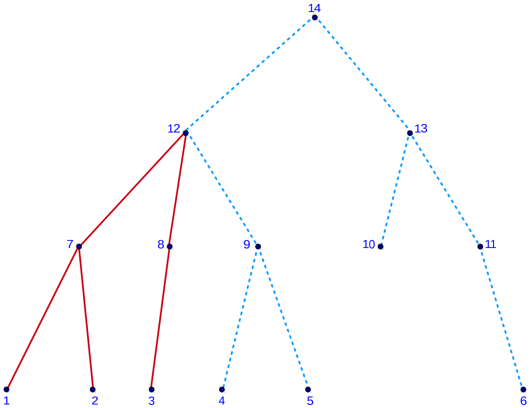
- the edge is not cut:
  the triplet (m_s, d_s, q_s) is transformed into

$$(m_s + 1,\ d_s + d_{newroot},\ q_s)$$

newroot = upper vertex of the edge under consideration

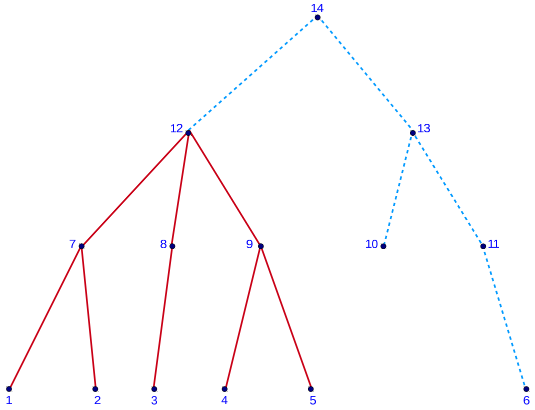
Algorithm: merging

Merging:
considers the effect of combining two at a time, in increasing order of labels, two subtrees rooted at the same vertex *v*.



Algorithm: merging

Merging:
considers the effect of combining two at a time, in increasing order of labels, two subtrees rooted at the same vertex v.



Algorithm: triplets update

Merging:

Two triplets are considered: (m_s, d_s, q_s) and (m'_s, d'_s, q'_s)
one in each of the two subtrees with the common root v,
and are transformed into

$$(m_s + m'_s,\ d_s + d'_s - d_v,\ q_s + q'_s)$$

Algorithm: dominance rules

- All pendent edges must belong to all optimal solutions

  $\Longrightarrow$ case cut edge not to be considered in extension

- Any cut edge induces a subtree not containing the root with maximum (local) modularity

  $\Longrightarrow$ in the list of triplets corresponding to cut edges only the triplet with maximum modularity needs to be kept

Algorithm: dominance rules

- Triplets may be dominated by other triplets.

  Let (m_1, d_1, q_1) and (m_2, d_2, q_2) two triplets in the same list:

  $$(m_1, d_1, q_1) \textit{ dominates } (m_2, d_2, q_2) \text{ if and only if}$$

  $$m_1 \geq m_2, \;\; d_1 \leq d_2 \;\; \text{and} \;\; q_1 \geq q_2 \text{ with at least one strict inequality.}$$

  Extension

  - edge is cut : in the following extension step there will be two corresponding triplets $\left(0, d_{newroot}, q_1 + \left(\frac{m_1}{m} - \left(\frac{d_1}{2m}\right)^2\right)\right)$ and $\left(0, d_{newroot}, q_2 + \left(\frac{m_2}{m} - \left(\frac{d_2}{2m}\right)^2\right)\right)$.
    $m_1 \geq m_2$, $d_1 \leq d_2$ and $q_1 \geq q_2$ with at least one strict inequality $\Rightarrow$ dominance
  - edge is not cut : there will be two corresponding triplets
    $(m_1 + 1, d_1 + d_{newroot}, q_1)$ and $(m_2 + 1, d_2 + d_{newroot}, q_2)$.
    $m_1 \geq m_2$, $d_1 \leq d_2$ and $q_1 \geq q_2$ with at least one strict inequality $\Rightarrow$ dominance

Algorithm: dominance rules

- Triplets may be dominated by other triplets.

  Let (m_1, d_1, q_1) and (m_2, d_2, q_2) two triplets in the same list:

  (m_1, d_1, q_1) *dominates* (m_2, d_2, q_2) if and only if

  $m_1 \geq m_2, \;\; d_1 \leq d_2$ and $q_1 \geq q_2$ with at least one strict inequality.

  Merging

  The two corresponding triplets will be:

  $(m_1 + m'_s, d_1 + d'_s - d_v, q_1 + q'_s)$ and $(m_2 + m'_s, d_2 + d'_s - d_v, q_2 + q'_s)$

  where: (m'_s, d'_s, q'_s) is a triplet of the list to merged to the list containing the two triplets

  d_v is the degree of the upper vertex of the current edge.

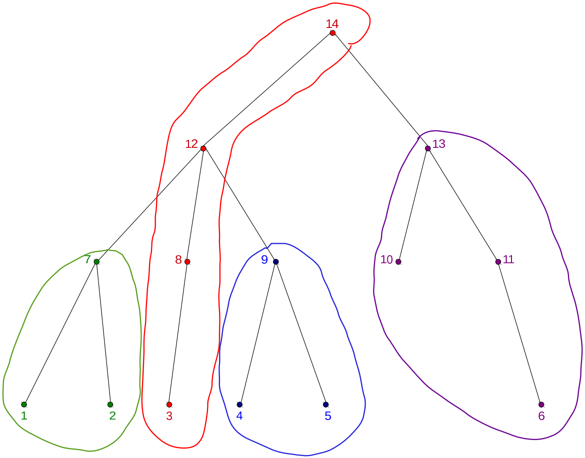
  $m_1 \geq m_2$, $d_1 \leq d_2$ and $q_1 \geq q_2$ with at least one strict inequality $\Rightarrow$ dominance

  By iteration $\Rightarrow$ the second triplet and its descendents are dominated by the first triplet and its descendent.

Algorithm: solution

The set of cut edges is complementary to the set of connected subtrees

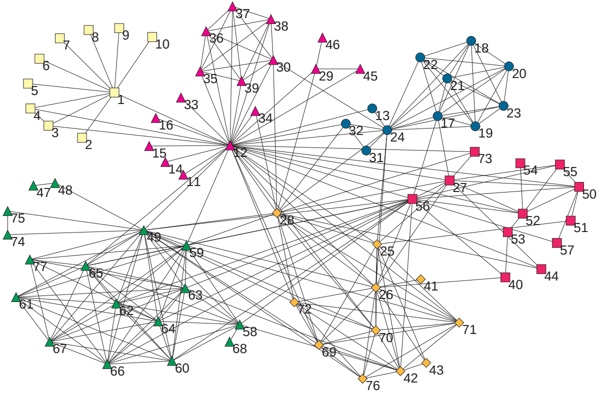
$\Rightarrow$ the optimal partition is given by the connected subtrees induced by all cut edges



Research directions

- Reformulations of the mathematical programming model for bipartitioning.
- Conditions which must be satisfied by all communities:
  combining a criterion for community evaluation with constraints on each community.
- Criteria other than modularity.

Thank you!


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77