Spectral Decomposition and Community Detection

Jérôme Galtier and Mikaila Toko-Worou

Spectral Properties of Complex Networks, Trente, July 23-27 2012

Graph decomposition

Let $G = (V, E)$ be a graph. We aim at decomposing G into some parts while :

- minimizing the number of edges in-between the parts,

orange™

Graph decomposition

Let $G = (V, E)$ be a graph. We aim at decomposing G into some parts while :

- minimizing the number of edges in-between the parts,
- balancing the sizes of the parts.

Two (main) criterions

The normalized cut *Ncut* (Shi and Malick, 1997) is inspired from the minimum cut and corrects it to produce balanced cuts.

Two (main) criterions

The normalized cut *Ncut* (Shi and Malick, 1997) is inspired from the minimum cut and corrects it to produce balanced cuts.

The modularity Q (Newman, 2003) mesures the quality of the partitioning using a comparison with a random model. The **quality** is simply the ratio of edges that are internal (inside a cluster).

orange™

Mathematical notations

Let A be the adjacency matrix of G, D the diagonal matrix of the degrees of the vertices.

Mathematical notations

Let A be the adjacency matrix of G, D the diagonal matrix of the degrees of the vertices.
Given a partitioning $\mathcal{C} = \{\mathcal{C}_1, \ldots \mathcal{C}_p\}$:

orange™

Mathematical notations

Let A be the adjacency matrix of G, D the diagonal matrix of the degrees of the vertices.

Given a partitioning $\mathcal{C} = \{\mathcal{C}_1, \ldots \mathcal{C}_p\}$:

$\mathcal{C}_i$ the set of vertices of i^{th} part, $i \in \{1, \ldots, p\}$,

Mathematical notations

Let A be the adjacency matrix of G, D the diagonal matrix of the degrees of the vertices.

Given a partitioning $\mathcal{C} = \{\mathcal{C}_1, \dots \mathcal{C}_p\}$:

$\mathcal{C}_i$ the set of vertices of i^{th} part, $i \in \{1, \dots, p\}$,

$\delta\mathcal{C}$ the set of edges in-between the parts,

Mathematical notations

Let A be the adjacency matrix of G, D the diagonal matrix of the degrees of the vertices.
Given a partitioning $\mathcal{C} = \{\mathcal{C}_1, \ldots \mathcal{C}_p\}$:

- $\mathcal{C}_i$ the set of vertices of i^{th} part, $i \in \{1, \ldots, p\}$,
- $\delta\mathcal{C}$ the set of edges in-between the parts,
- $\delta\mathcal{C}_i$ the set of edges in-between the part $\mathcal{C}_i$ and others,

Mathematical notations

Let A be the adjacency matrix of G, D the diagonal matrix of the degrees of the vertices.
Given a partitioning $\mathcal{C} = \{\mathcal{C}_1, \ldots \mathcal{C}_p\}$:

- $\mathcal{C}_i$ the set of vertices of i^{th} part, $i \in \{1, \ldots, p\}$,
- $\delta\mathcal{C}$ the set of edges in-between the parts,
- $\delta\mathcal{C}_i$ the set of edges in-between the part $\mathcal{C}_i$ and others,
- k_i the size of $\mathcal{C}_i$, given by

$$k_i = \frac{1}{2|E|} \sum_{v \in \mathcal{C}_i} deg(v).$$

Mathematical notations

Let A be the adjacency matrix of G, D the diagonal matrix of the degrees of the vertices.

Given a partitioning $\mathcal{C} = \{\mathcal{C}_1, \ldots \mathcal{C}_p\}$:

- $\mathcal{C}_i$ the set of vertices of i^{th} part, $i \in \{1, \ldots, p\}$,
- $\delta\mathcal{C}$ the set of edges in-between the parts,
- $\delta\mathcal{C}_i$ the set of edges in-between the part $\mathcal{C}_i$ and others,
- k_i the size of $\mathcal{C}_i$, given by

$$k_i = \frac{1}{2|E|} \sum_{v \in \mathcal{C}_i} deg(v).$$

Note that we have $\sum_{i \in \{1,\ldots,p\}} k_i = 1$ and $\sum_{i \in \{1,\ldots,p\}} |\delta\mathcal{C}_i| = 2|\delta\mathcal{C}|$.

What is NCut

Computing the minimum cut is very easy (it minimizes $|\delta\mathcal{C}|$), but usually isolates small parts of the graph

What is NCut

Computing the minimum cut is very easy (it minimizes $|\delta \mathcal{C}|$), but usually isolates small parts of the graph

The **minimum NCut** consists in minimizing :

$$\sum_{i=1}^{i=p} \frac{|\delta \mathcal{C}_i|}{k_i}.$$

What is modularity

$Random(E)$ is a random set of edges giving the same vertex degree as G. We consider a clustering $\mathcal{C}$ (a partition of V).

What is modularity

$Random(E)$ is a random set of edges giving the same vertex degree as G. We consider a clustering $\mathcal{C}$ (a partition of V). The intuitive definition of Newman is :

- ρ is the probability that an edge of E taken at random is in a cluster $\mathcal{C}_i$ of $\mathcal{C}$,

What is modularity

Random(E) is a random set of edges giving the same vertex degree as G. We consider a clustering $\mathcal{C}$ (a partition of V). The intuitive definition of Newman is :

- ρ is the probability that an edge of E taken at random is in a cluster $\mathcal{C}_i$ of $\mathcal{C}$,
- ρ' is the probability that an edge of Random(E) taken at random is in a cluster $\mathcal{C}_i$ of $\mathcal{C}$,

What is modularity

Random(E) is a random set of edges giving the same vertex degree as G. We consider a clustering $\mathcal{C}$ (a partition of V). The intuitive definition of Newman is :

- ρ is the probability that an edge of E taken at random is in a cluster $\mathcal{C}_i$ of $\mathcal{C}$,
- ρ' is the probability that an edge of Random(E) taken at random is in a cluster $\mathcal{C}_i$ of $\mathcal{C}$,

The modularity $Q(C)$ is $\rho - \rho'$

Formulas for the two criterions

The normalized cut *Ncut*

$$Ncut(\mathcal{C}) = \sum_{i=1}^{i=p} \frac{|\delta \mathcal{C}_i|}{k_i}.$$

Formulas for the two criterions

The normalized cut *Ncut*

$$Ncut(\mathcal{C}) = \sum_{i=1}^{i=p} \frac{|\delta \mathcal{C}_i|}{k_i}.$$

The modularity *Q*

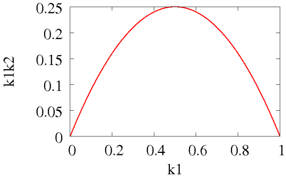
$$Q(\mathcal{C}) = 1 - \frac{|\delta \mathcal{C}|}{|E|} - \sum_{i=1}^{i=p} {k_i}^2.$$

Formula for a bipartition (p=2)

We have $k_1 + k_2 = 1$ and we derive the following expressions in $k_1 k_2$:

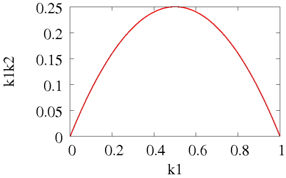
Formula for a bipartition (p=2)

We have $k_1 + k_2 = 1$ and we derive the following expressions in $k_1 k_2$:



Formula for a bipartition (p=2)

We have $k_1 + k_2 = 1$ and we derive the following expressions in $k_1 k_2$:

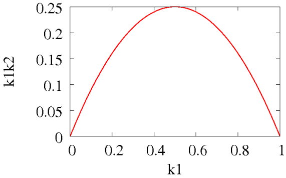


Normalized cut

$$Ncut(\mathcal{C}) = \frac{|\delta \mathcal{C}|}{k_1 k_2}.$$

Formula for a bipartition (p=2)

We have $k_1 + k_2 = 1$ and we derive the following expressions in $k_1 k_2$:



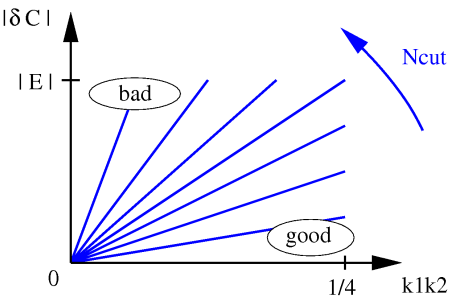
Normalized cut

$$Ncut(\mathcal{C}) = \frac{|\delta \mathcal{C}|}{k_1 k_2}.$$

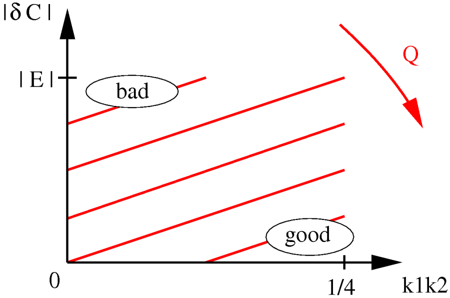
Modularity

$$Q(\mathcal{C}) = 2k_1 k_2 - \frac{|\delta \mathcal{C}|}{|E|}.$$

Insight for a bipartition

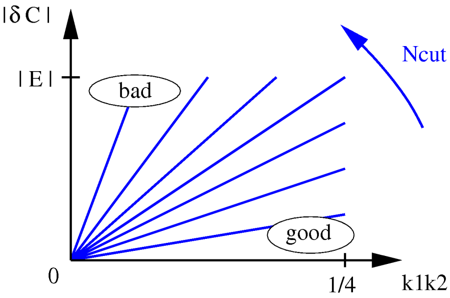


minimizing *Ncut*

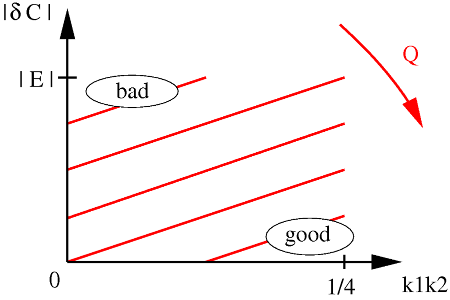


maximizing *Q*

Insight for a bipartition



minimizing *Ncut*



maximizing *Q*

→ the two criterias are extremely similar !

Further analysis for the NCut using matrix form

ξ is the n indicator vector ($\xi_u = 1$ if node u is in C_1 and -1, otherwise)

$$Ncut(\mathcal{C}) = 2|\delta\mathcal{C}|\left(\frac{1}{k_1} + \frac{1}{k_2}\right)$$

Further analysis for the NCut using matrix form

ξ is the n indicator vector ($\xi_u = 1$ if node u is in C_1 and -1, otherwise)

$$Ncut(\mathcal{C}) = 2|\delta\mathcal{C}|\left(\frac{1}{k_1}+\frac{1}{k_2}\right) = \sum_{i,j,\xi_i\xi_j<0} -A_{ij}\xi_i\xi_j\left(\frac{1}{k_1}+\frac{1}{k_2}\right)$$

orange™

Further analysis for the NCut using matrix form

ξ is the n indicator vector ($\xi_u = 1$ if node u is in C_1 and -1, otherwise)

$$Ncut(\mathcal{C}) = 2|\delta\mathcal{C}|\left(\frac{1}{k_1}+\frac{1}{k_2}\right) = \sum_{i,j,\xi_i\xi_j<0} -A_{ij}\xi_i\xi_j\left(\frac{1}{k_1}+\frac{1}{k_2}\right)$$

It gives

$$Ncut(\mathcal{C}) = \frac{[(\mathbf{1}+\xi)-b(\mathbf{1}-\xi)]^t(D-A)[(\mathbf{1}+\xi)-b(\mathbf{1}-\xi)]}{b\mathbf{1}^t D\mathbf{1}},$$

$$\text{with } b = \frac{k_1}{1-k_1}.$$

orange™

Further analysis for the NCut using matrix form

ξ is the n indicator vector ($\xi_u = 1$ if node u is in C_1 and -1, otherwise)

$$Ncut(\mathcal{C}) = 2|\delta\mathcal{C}|\left(\frac{1}{k_1}+\frac{1}{k_2}\right) = \sum_{i,j,\xi_i\xi_j<0} -A_{ij}\xi_i\xi_j\left(\frac{1}{k_1}+\frac{1}{k_2}\right)$$

It gives $$Ncut(\mathcal{C}) = \frac{[(\mathbf{1}+\xi)-b(\mathbf{1}-\xi)]^t(D-A)[(\mathbf{1}+\xi)-b(\mathbf{1}-\xi)]}{b\mathbf{1}^tD\mathbf{1}},$$ $$with\ b = \frac{k_1}{1-k_1}.$$

By setting $Y = (\mathbf{1}+\xi) - b(\mathbf{1}-\xi)$, we have $Y^TD\mathbf{1} = 0$ and $b\mathbf{1}^TD\mathbf{1} = Y^TDY$.

$\min_\xi Ncut(\xi)$ st : $\xi \in \{-1,1\}^n$	$\Leftrightarrow$	$\min_Y \frac{Y^T(D-A)Y}{Y^TDY}$ st : $Y^TD\mathbf{1} = 0$ and $Yu = \{-b,1\}\ \forall u \in V$, depending on $u \in C_1$, or $u \in C_2$.

What it means. . .

- the expression above is the **Rayleigh** quotient.

What it means...

- the expression above is the **Rayleigh** quotient.
- by a relaxation, we consider Y can take real values in $[-b, 1]$, the solution for minimizing the Rayleigh quotient is given by solving the generalized eigenvalue system $(D - A)Y = \lambda DY \textit{ subject to } Y^T D\mathbf{1} = 0$.

What it means…

- the expression above is the **Rayleigh** quotient.
- by a relaxation, we consider Y can take real values in $[-b, 1]$, the solution for minimizing the Rayleigh quotient is given by solving the generalized eigenvalue system $(D-A)Y = \lambda DY \textit{ subject to } Y^T D\mathbf{1} = 0$.
- the solution is the eigenvector corresponding to the **second smallest eigenvalue**.

orange™

What about modularity ?

Denote $X = {}^t YY$ and suppose Y is a $1 \times n$ vector defined as follows (note that since $X_{u,u} = 1$, we have $y_u^2 = 1$) :

$$\begin{cases} y_u = 1 & \text{if } u \in \mathcal{C}_1 \\ y_u = -1 & \text{if } u \in \mathcal{C}_2 \end{cases}$$

What about modularity ?

Denote $X =^t YY$ and suppose Y is a $1 \times n$ vector defined as follows (note that since $X_{u,u} = 1$, we have $y_u^2 = 1$) :

$$\begin{cases} y_u = 1 & \text{if } u \in \mathcal{C}_1 \\ y_u = -1 & \text{if } u \in \mathcal{C}_2 \end{cases}$$

$$(A - \frac{1}{2|E|} DJD) \cdot X = -4|\delta\mathcal{C}| + 8|E| k_1 k_2.$$

both spectral relaxations put together

Let J the square matrix filled with 1's.

both spectral relaxations put together

Let J the square matrix filled with 1's.

NCut	*Modularity*
$\max(D^{-1/2}AD^{-1/2} - I) \cdot X$	$\max(A - \frac{1}{2\|E\|}DJD) \cdot X$
s.t. $X \cdot (D^{1/2}JD^{1/2}) = 0$	s.t. $X_{v,v} = 1 \ \forall v \in V$
$X \cdot I = \|V\|$	
$X \succeq 0$	$X \succeq 0$

cutting with eigenvalues : numerical results for Q

	karate	Arxiv
CNM	0.38	0.772
PL	0.42	0.757
WT	0.42	0.767
Louvain	0.42	0.813
SpecMod	0.42	0.772
SpecMod with refinement	0.42	0.801

tools when eigenvalues are not available

A matrix X is semidefinite positive if and only if there exists a matrix Y such that

$$X =^t YY.$$

Let Y_v, $v \in V$, be the set of columns of Y. Each of them is a vector of $\mathbb{R}^n$.

tools when eigenvalues are not available

A matrix X is semidefinite positive if and only if there exists a matrix Y such that

$$X =^t YY.$$

Let Y_v, $v \in V$, be the set of columns of Y. Each of them is a vector of $\mathbb{R}^n$.

We have the following properties

- $\forall v \in V \qquad X_{v,v} = 1 \Leftrightarrow |Y_v| = 1,$

tools when eigenvalues are not available

A matrix X is semidefinite positive if and only if there exists a matrix Y such that

$$X =^t YY.$$

Let Y_v, $v \in V$, be the set of columns of Y. Each of them is a vector of $\mathbb{R}^n$.

We have the following properties

- $\forall v \in V \qquad X_{v,v} = 1 \Leftrightarrow |Y_v| = 1,$
- $\forall L \in \mathbb{R}^n \qquad X \cdot ({}^t LL) = 0 \Leftrightarrow \forall v \in V \quad Y_v \cdot L = 0,$

tools when eigenvalues are not available

A matrix X is semidefinite positive if and only if there exists a matrix Y such that

$$X =^t YY.$$

Let Y_v, $v \in V$, be the set of columns of Y. Each of them is a vector of $\mathbb{R}^n$.

We have the following properties

- $\forall v \in V \qquad X_{v,v} = 1 \Leftrightarrow |Y_v| = 1,$
- $\forall L \in \mathbb{R}^n \qquad X \cdot ({}^t LL) = 0 \Leftrightarrow \forall v \in V \quad Y_v \cdot L = 0,$
- $$\forall W \in \mathbb{R}^{n,n} \qquad X \cdot W = \sum_{u,v \in V} Y_u Y_v W_{u,v},$$

tools when eigenvalues are not available

A matrix X is semidefinite positive if and only if there exists a matrix Y such that

$$X =^t YY.$$

Let Y_v, $v \in V$, be the set of columns of Y. Each of them is a vector of $\mathbb{R}^n$.

We have the following properties

- $\forall v \in V \qquad X_{v,v} = 1 \Leftrightarrow |Y_v| = 1,$
- $\forall L \in \mathbb{R}^n \qquad X \cdot ({}^tLL) = 0 \Leftrightarrow \forall v \in V \quad Y_v \cdot L = 0,$
- $$\forall W \in \mathbb{R}^{n,n} \qquad X \cdot W = \sum_{u,v \in V} Y_u Y_v W_{u,v},$$
- $$X \cdot I = \sum_{v \in V} |Y_v|^2.$$

using KKT conditions !

we associate to each vertex u a point Y_u in $\mathbb{R}^d$ with d smaller than n.

orange™

using KKT conditions !

we associate to each vertex u a point Y_u in $\mathbb{R}^d$ with d smaller than n.

problem : $\boxed{\text{maximize} \sum_{\{u,v\}\in E} W_{u,v} Y_u \cdot Y_v.}$

the algorithm proposed

the algorithm used is **incremental**.

the 𝔞𝔩𝔤𝔬𝔯𝔦𝔱𝔥𝔪 proposed

the algorithm used is **incremental**.

we update each vertex u as follows :

the $\mathfrak{algorithm}$ proposed

the algorithm used is **incremental**.

we update each vertex u as follows :

- compute the value $Z_u := \sum_{v \in V - \{u\}} W_{u,v} Y_v$,

the algorithm proposed

the algorithm used is **incremental**.

we update each vertex u as follows :

- compute the value $Z_u := \sum_{v \in V - \{u\}} W_{u,v} Y_v$,
- if $Z_u \neq 0$ and we have to keep $|Y_u|$ constant, we perform the operation $Y_u := -Z_u |Y_u| / |Z_u|$,

the algorithm proposed

the algorithm used is **incremental**.

we update each vertex u as follows :

- compute the value $Z_u := \sum_{v \in V - \{u\}} W_{u,v} Y_v$,
- if $Z_u \neq 0$ and we have to keep $|Y_u|$ constant, we perform the operation $Y_u := -Z_u |Y_u| / |Z_u|$,
- if $Z_u = 0$, choose Y_u at random.

the algorithm proposed

the algorithm used is **incremental**.

we update each vertex u as follows :

- compute the value $Z_u := \sum_{v \in V - \{u\}} W_{u,v} Y_v$,
- if $Z_u \neq 0$ and we have to keep $|Y_u|$ constant, we perform the operation $Y_u := -Z_u |Y_u| / |Z_u|$,
- if $Z_u = 0$, choose Y_u at random.

$\rightarrow$ we necessarily increase the value $X \cdot W$.

the algorithm proposed

the algorithm used is **incremental**.

we update each vertex u as follows :

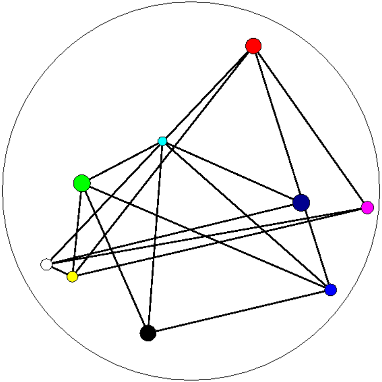
- compute the value $Z_u := \sum_{v \in V - \{u\}} W_{u,v} Y_v$,
- if $Z_u \neq 0$ and we have to keep $|Y_u|$ constant, we perform the operation $Y_u := -Z_u |Y_u| / |Z_u|$,
- if $Z_u = 0$, choose Y_u at random.

$\rightarrow$ we necessarily increase the value $X \cdot W$.

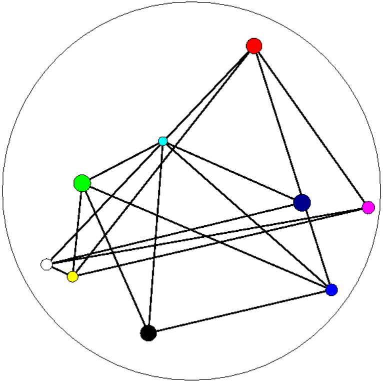
In practice, for $d = 3$, **3 min** are required to update 97.10^6 **nodes**.

compared methods

Ncut (using Lagrange relaxation)

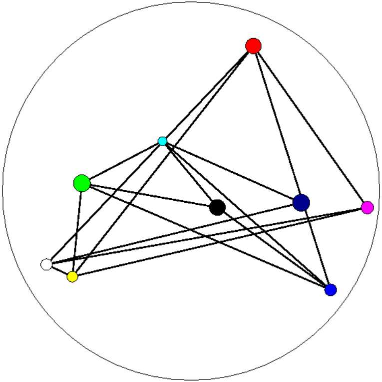


Modularity

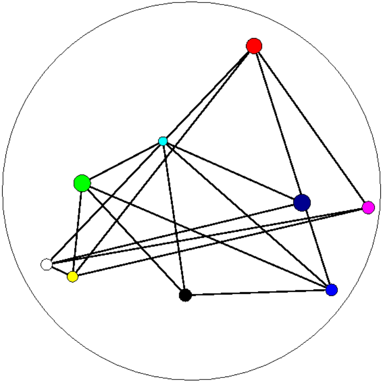


compared methods

Ncut (using Lagrange relaxation)

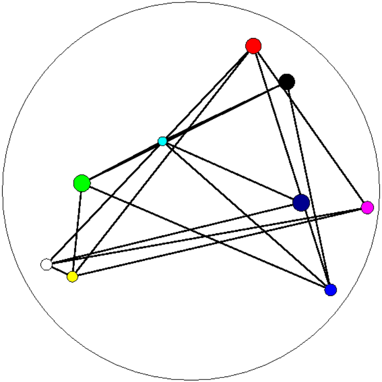


Modularity

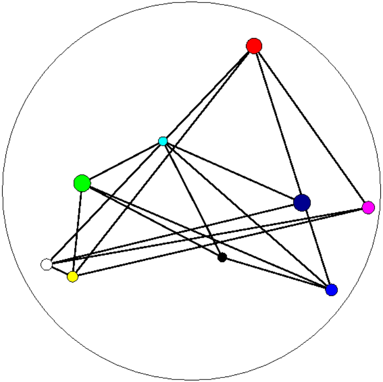


compared methods

Ncut (using Lagrange relaxation)

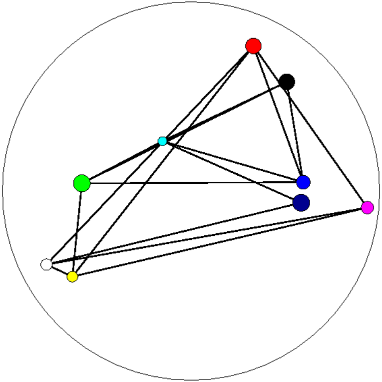


Modularity

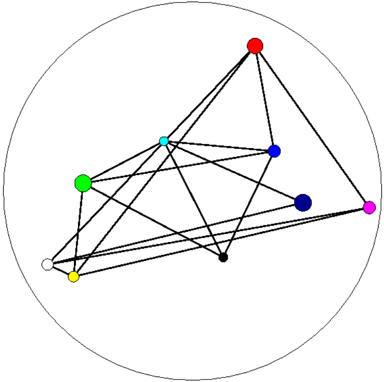


compared methods

Ncut (using Lagrange relaxation)

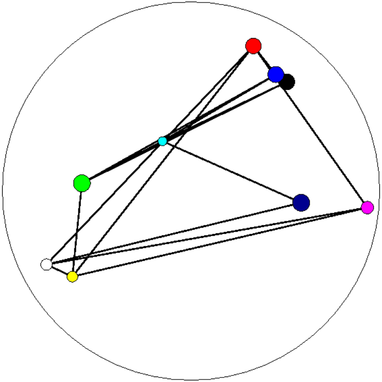


Modularity

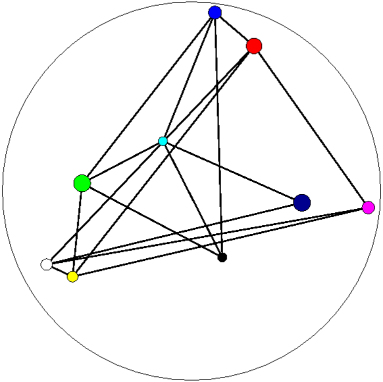


compared methods

Ncut (using Lagrange relaxation)

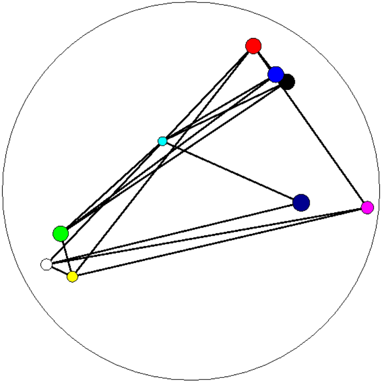


Modularity

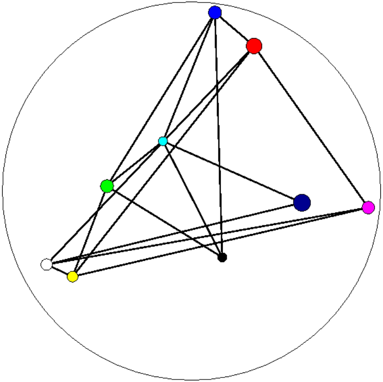


compared methods

Ncut (using Lagrange relaxation)

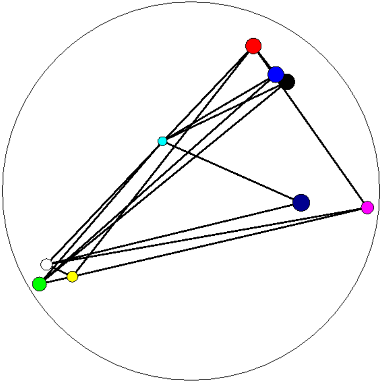


Modularity

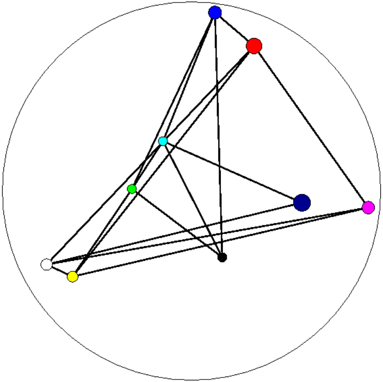


compared methods

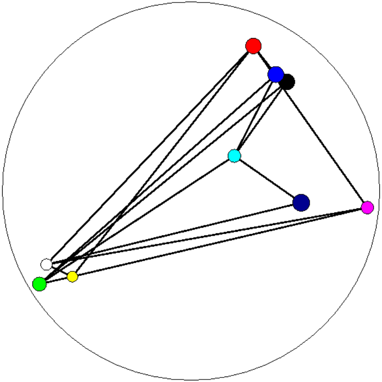
Ncut (using Lagrange relaxation)

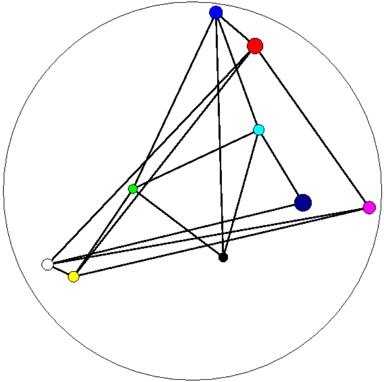


Modularity



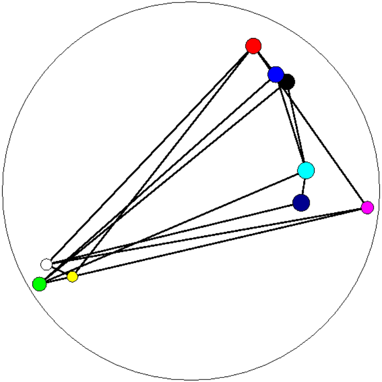
compared methods



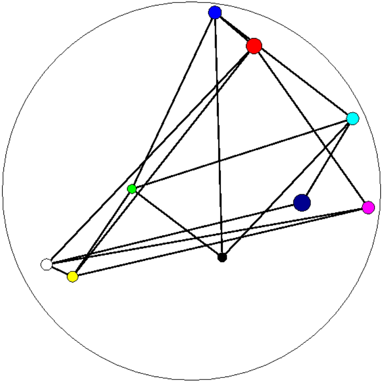


compared methods

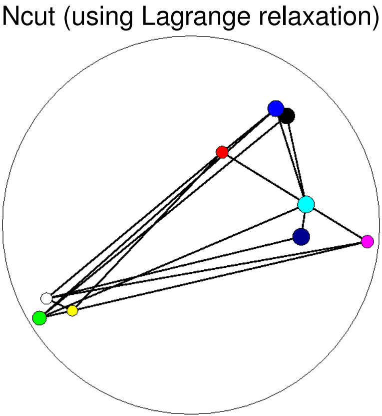
Ncut (using Lagrange relaxation)

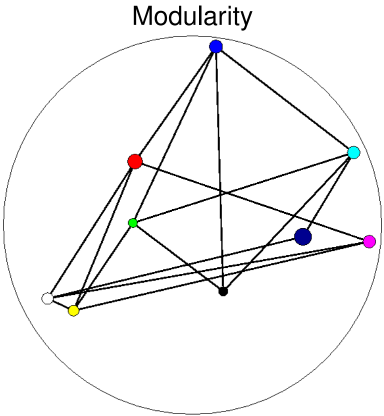


Modularity



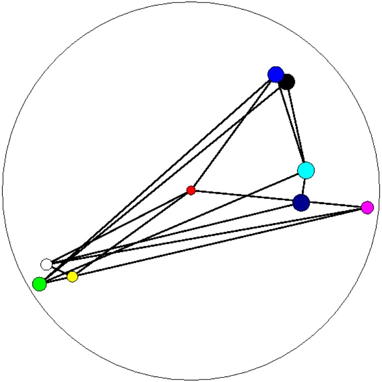
compared methods



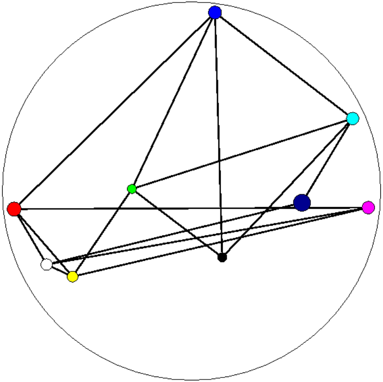


compared methods

Ncut (using Lagrange relaxation)

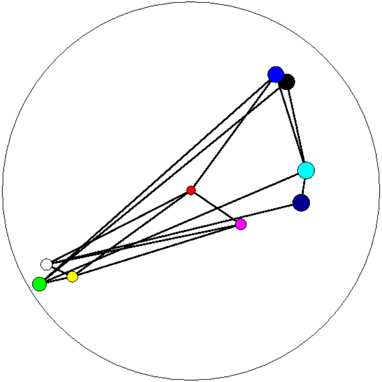


Modularity

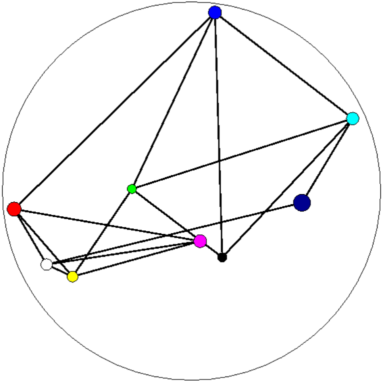


compared methods

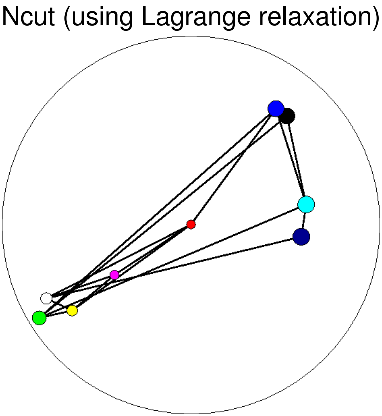
Ncut (using Lagrange relaxation)

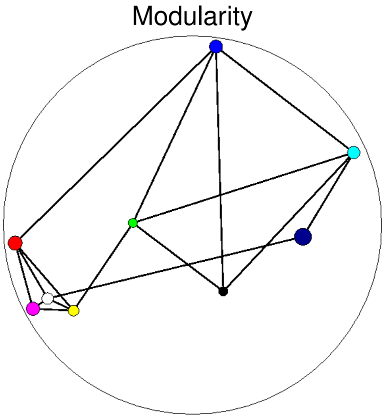


Modularity



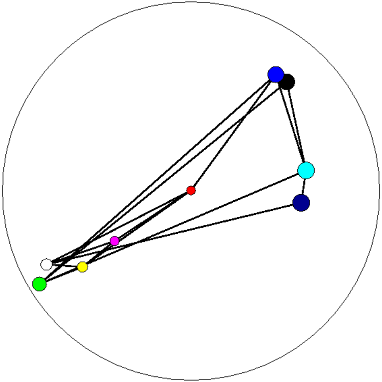
compared methods



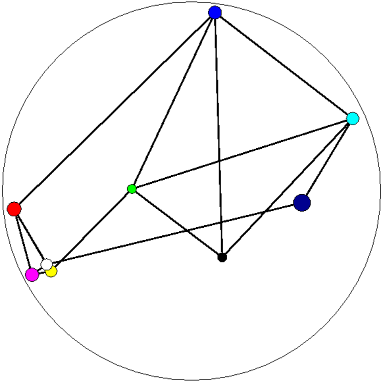


compared methods

Ncut (using Lagrange relaxation)

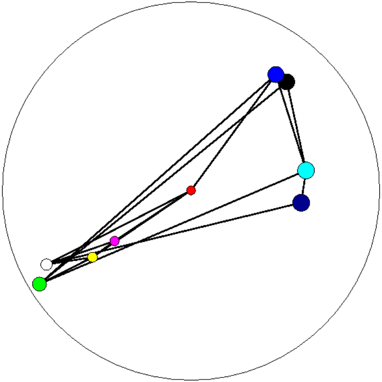


Modularity

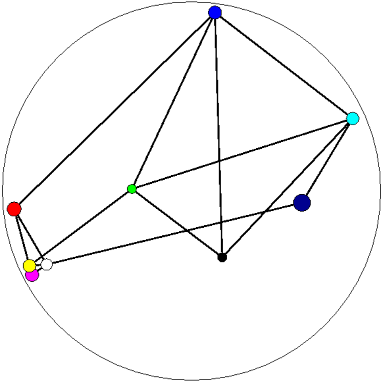


compared methods

Ncut (using Lagrange relaxation)

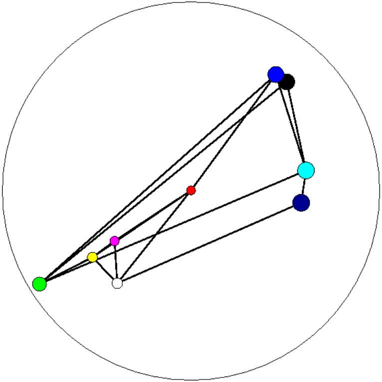


Modularity

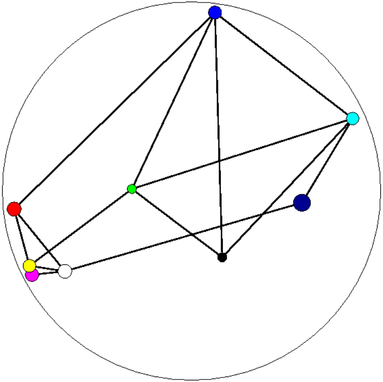


compared methods

Ncut (using Lagrange relaxation)

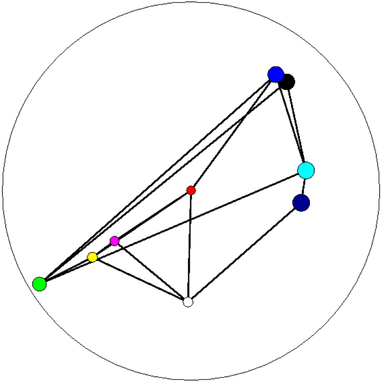


Modularity

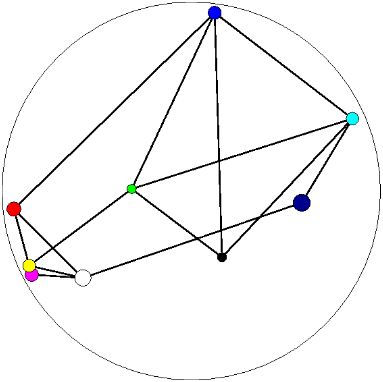


compared methods

Ncut (using Lagrange relaxation)

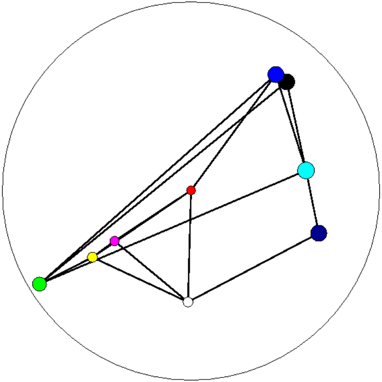


Modularity

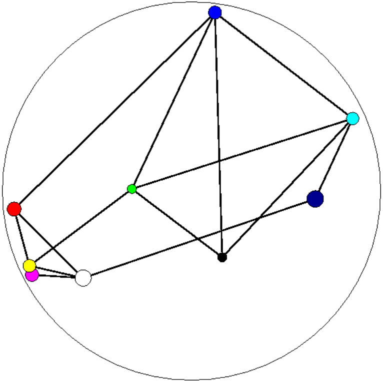


compared methods

Ncut (using Lagrange relaxation)

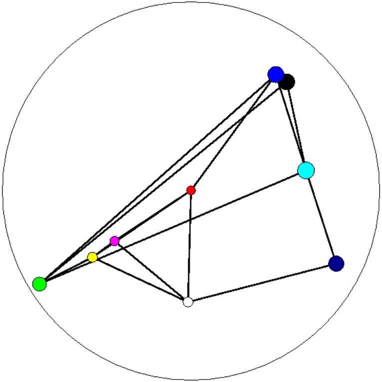


Modularity

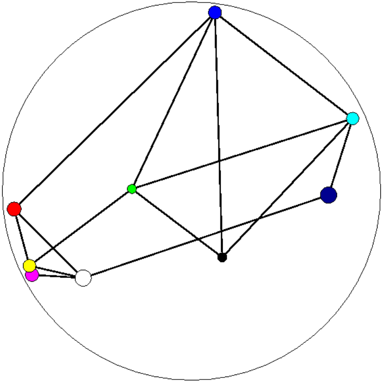


compared methods

Ncut (using Lagrange relaxation)

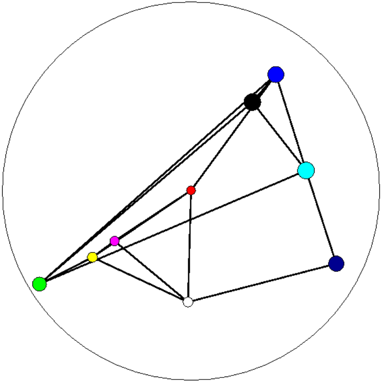


Modularity

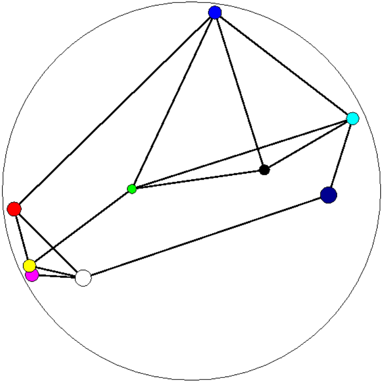


compared methods

Ncut (using Lagrange relaxation)

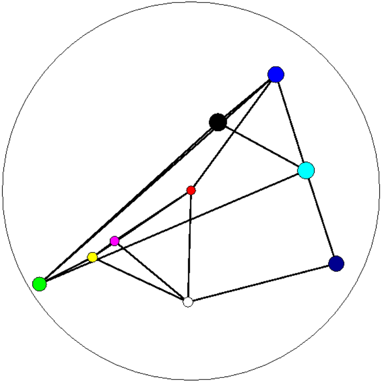


Modularity

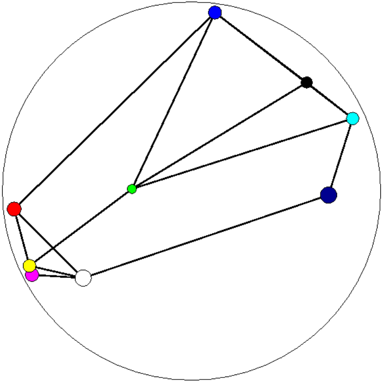


compared methods

Ncut (using Lagrange relaxation)

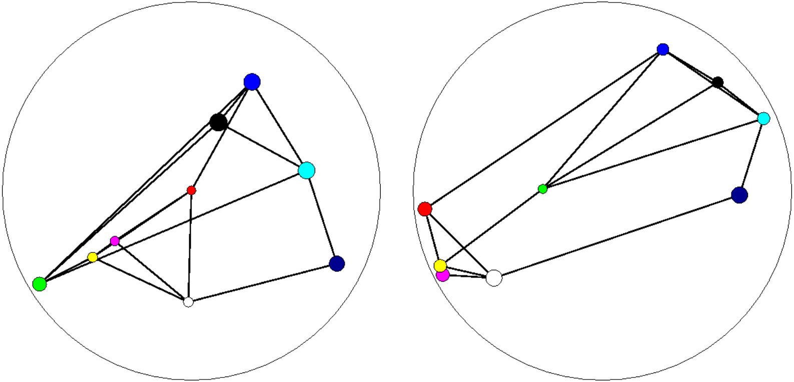


Modularity



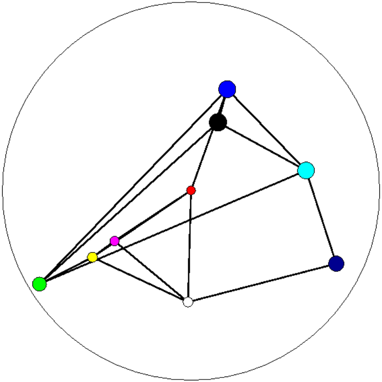
compared methods

Ncut (using Lagrange relaxation) — Modularity

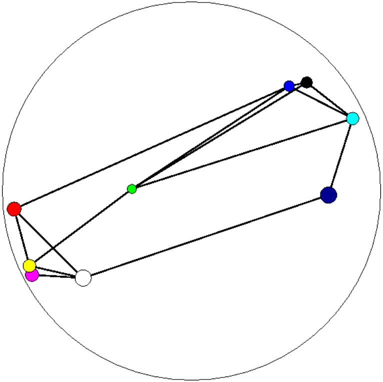


compared methods

Ncut (using Lagrange relaxation)

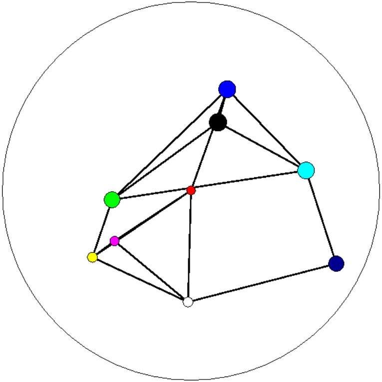


Modularity

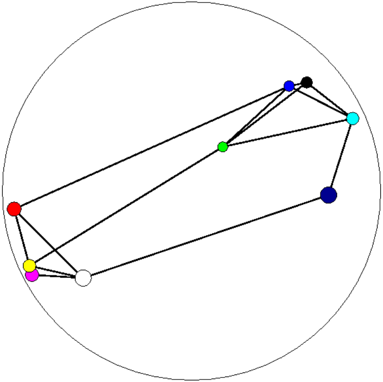


compared methods

Ncut (using Lagrange relaxation)

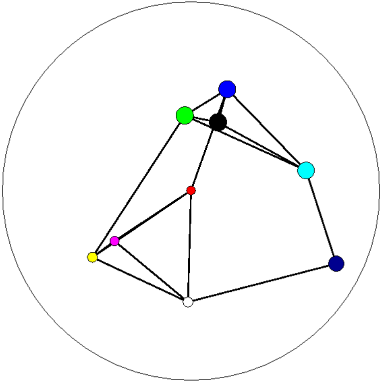


Modularity

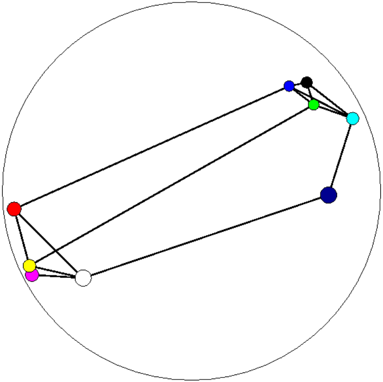


compared methods

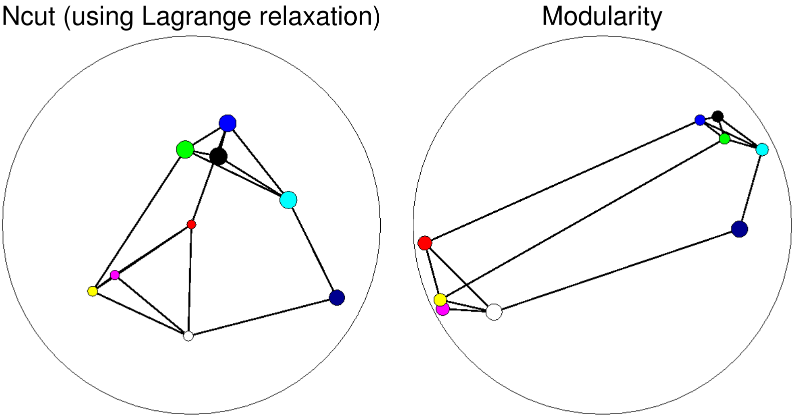
Ncut (using Lagrange relaxation)



Modularity

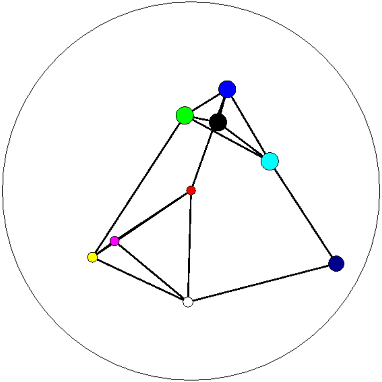


compared methods

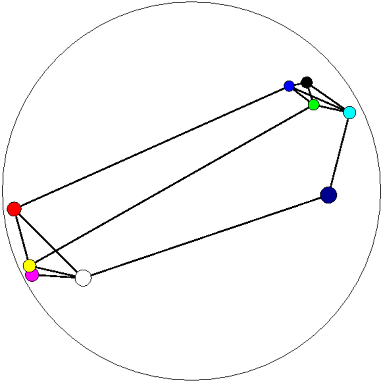


compared methods

Ncut (using Lagrange relaxation)



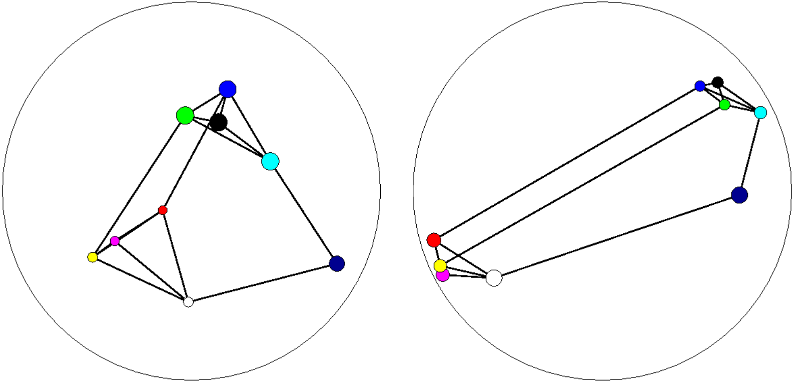
Modularity



compared methods

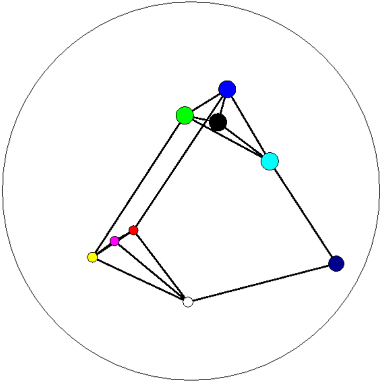
Ncut (using Lagrange relaxation)

Modularity

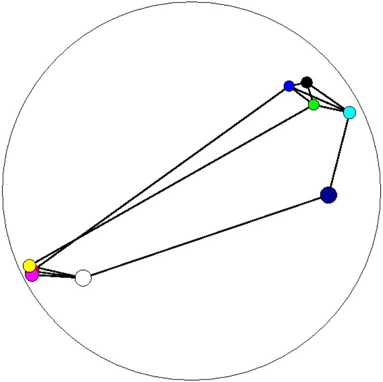


compared methods

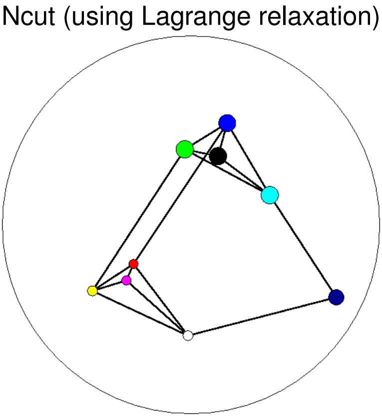
Ncut (using Lagrange relaxation)

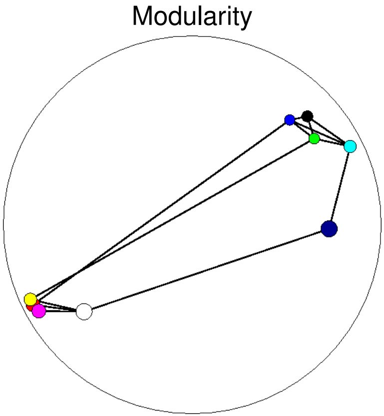


Modularity



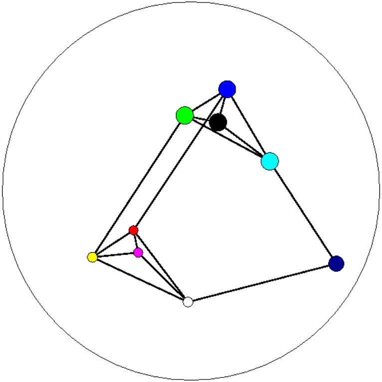
compared methods



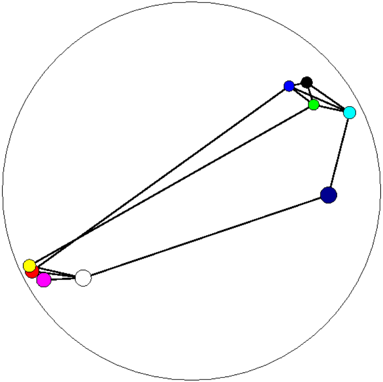


compared methods

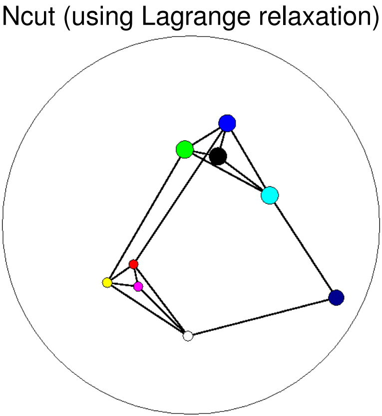
Ncut (using Lagrange relaxation)

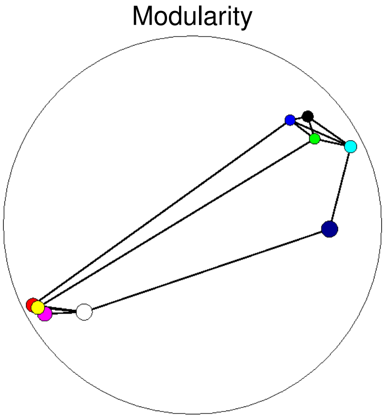


Modularity

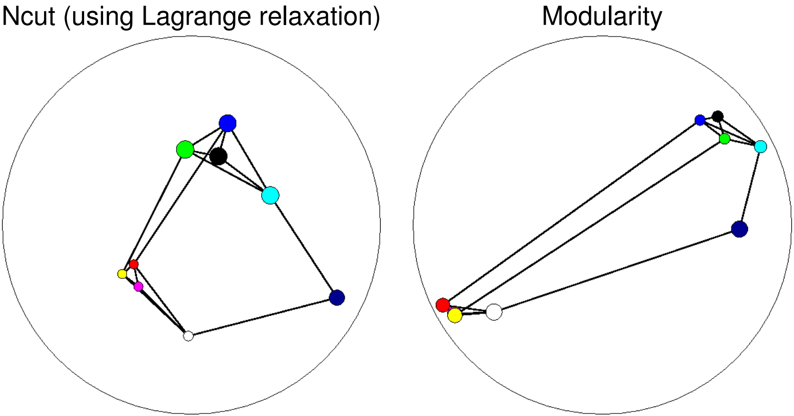


compared methods



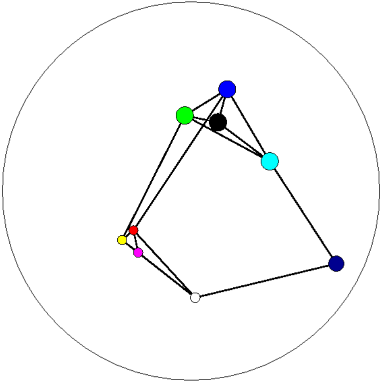


compared methods

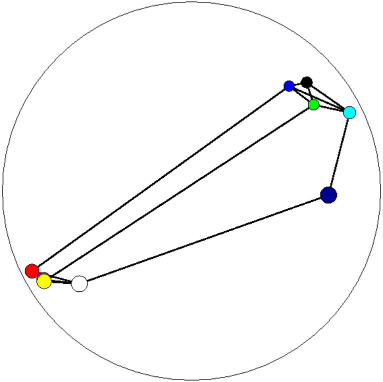


compared methods

Ncut (using Lagrange relaxation)

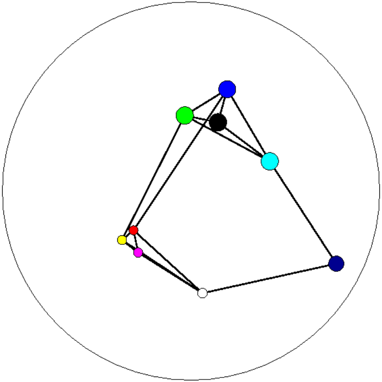


Modularity

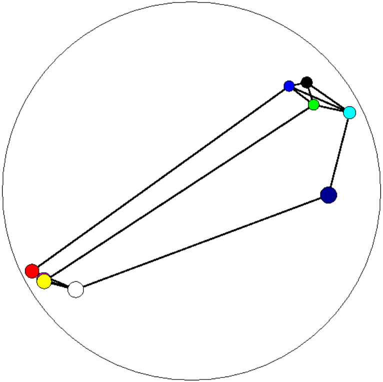


compared methods

Ncut (using Lagrange relaxation)

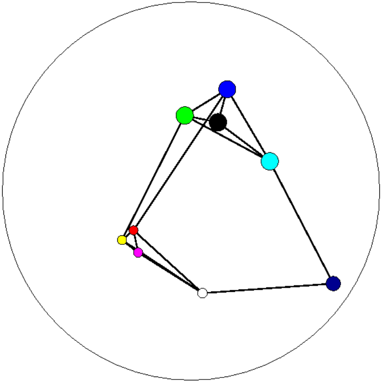


Modularity

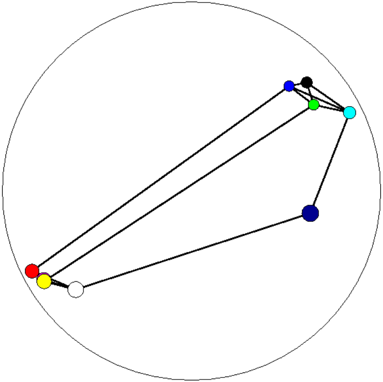


compared methods

Ncut (using Lagrange relaxation)

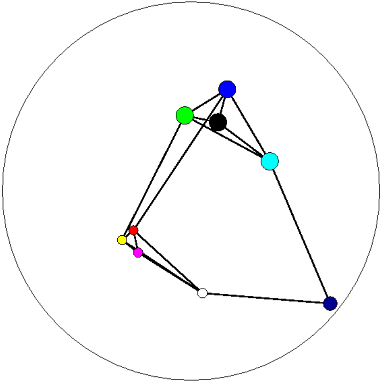


Modularity

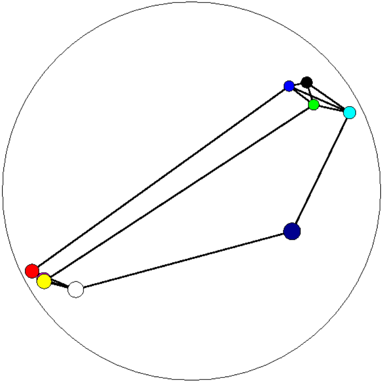


compared methods

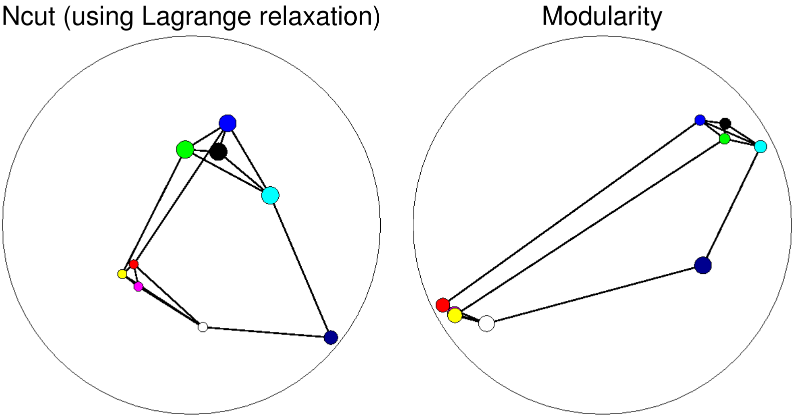
Ncut (using Lagrange relaxation)



Modularity

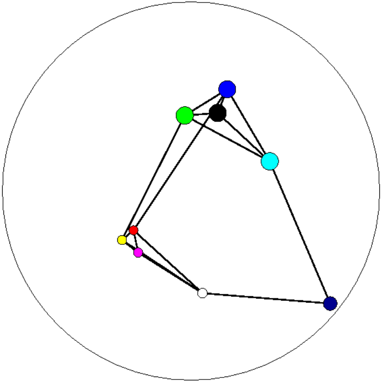


compared methods

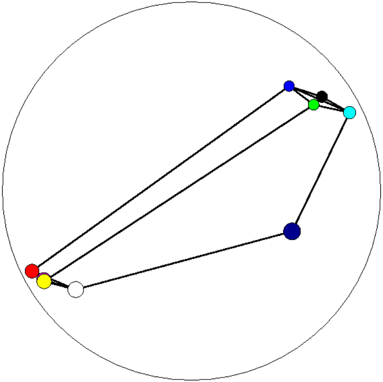


compared methods

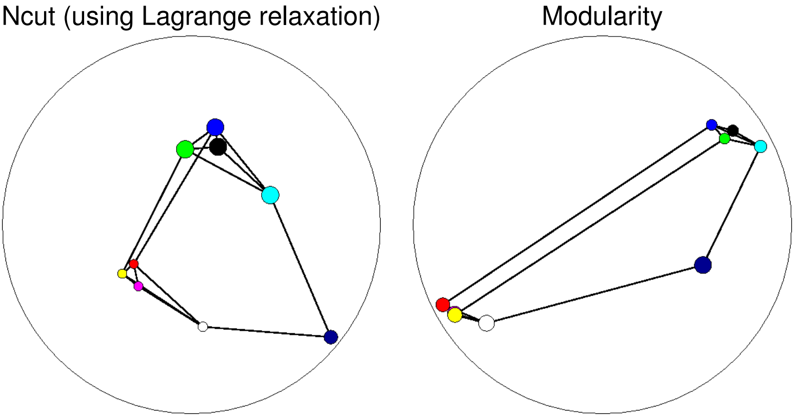
Ncut (using Lagrange relaxation)



Modularity

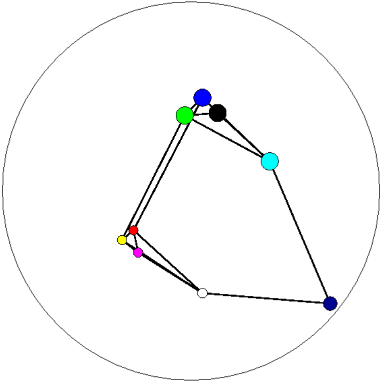


compared methods

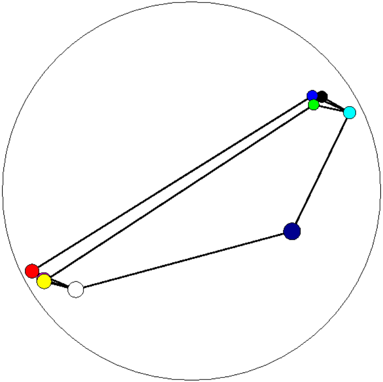


compared methods

Ncut (using Lagrange relaxation)

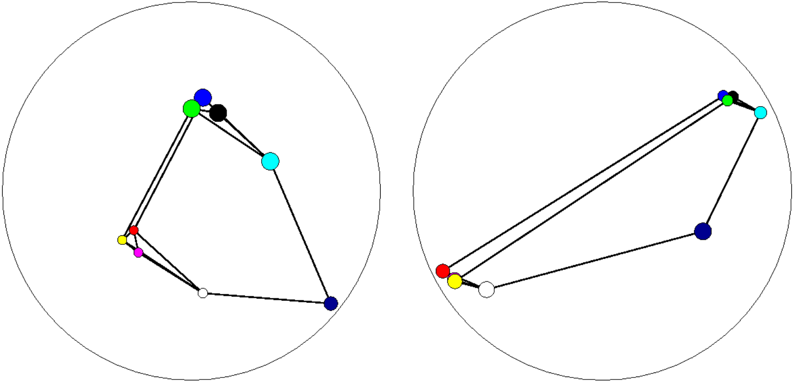


Modularity



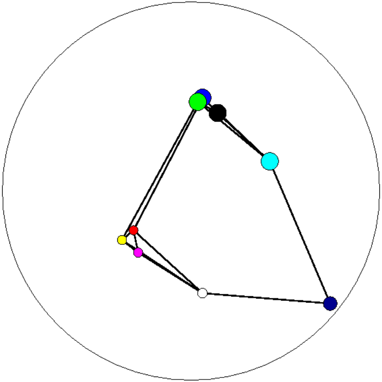
compared methods

Ncut (using Lagrange relaxation)   Modularity

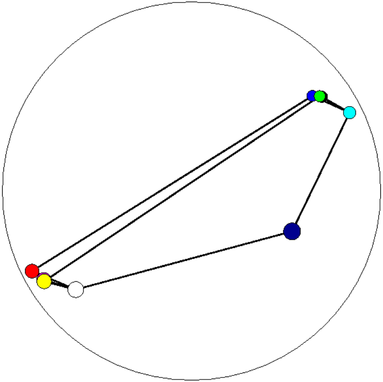


compared methods

Ncut (using Lagrange relaxation)

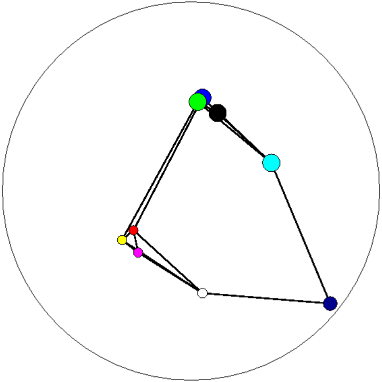


Modularity

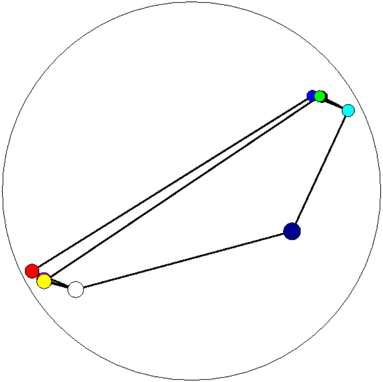


compared methods

Ncut (using Lagrange relaxation)

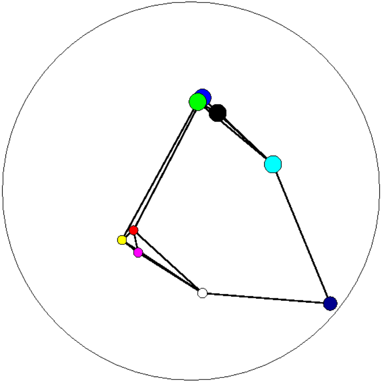


Modularity

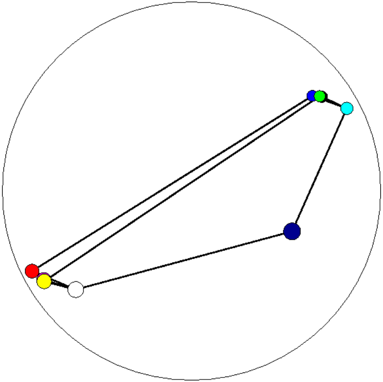


compared methods

Ncut (using Lagrange relaxation)

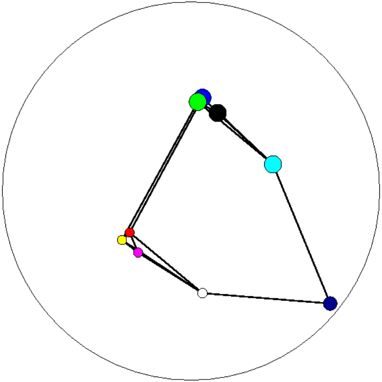


Modularity

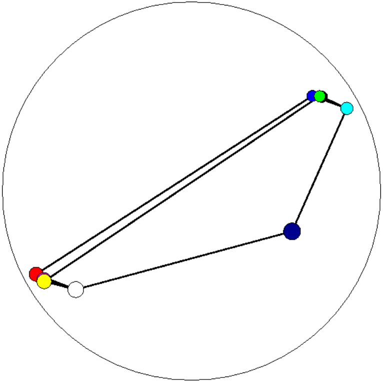


compared methods

Ncut (using Lagrange relaxation)

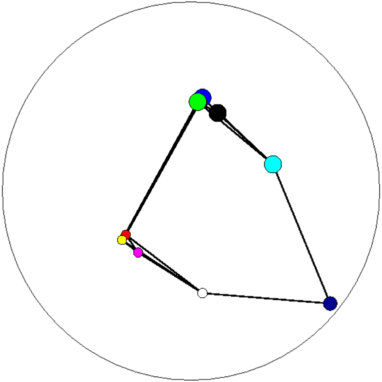


Modularity

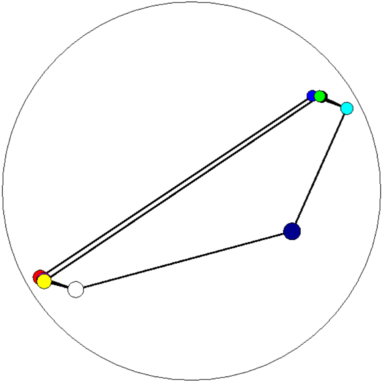


compared methods

Ncut (using Lagrange relaxation)

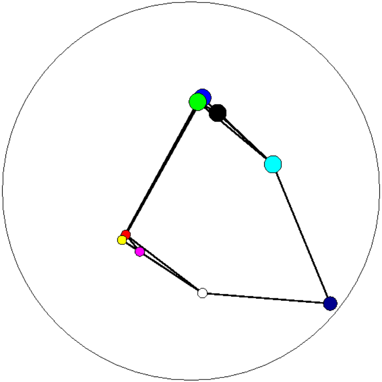


Modularity

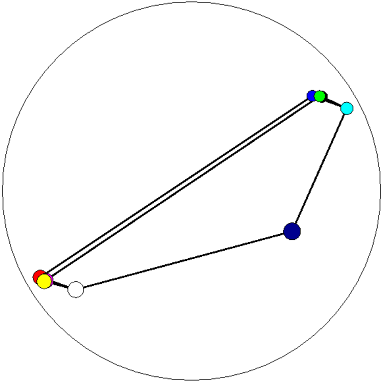


compared methods

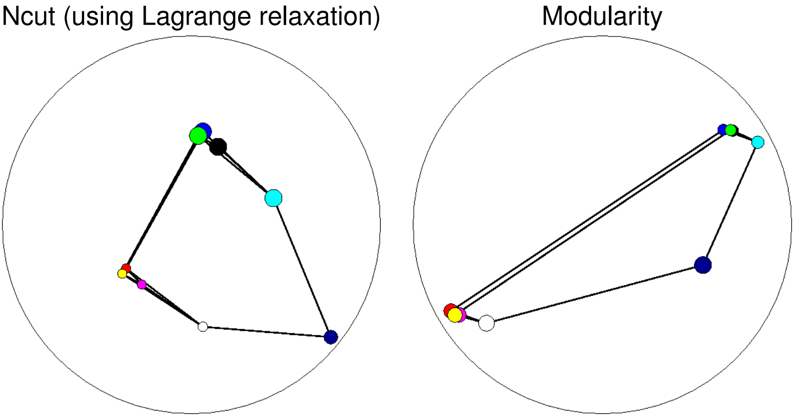
Ncut (using Lagrange relaxation)



Modularity

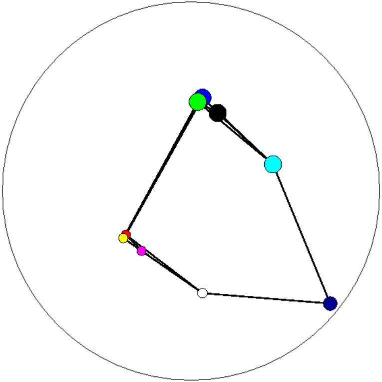


compared methods

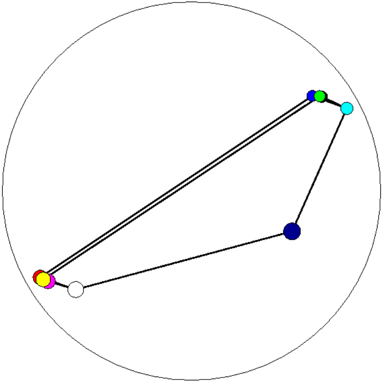


compared methods

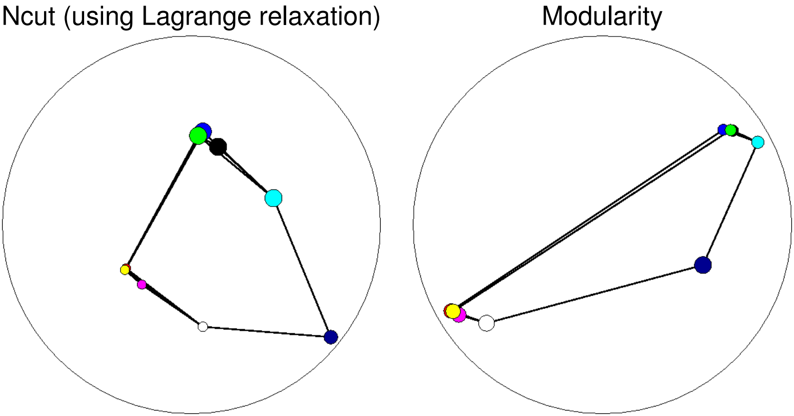
Ncut (using Lagrange relaxation)



Modularity

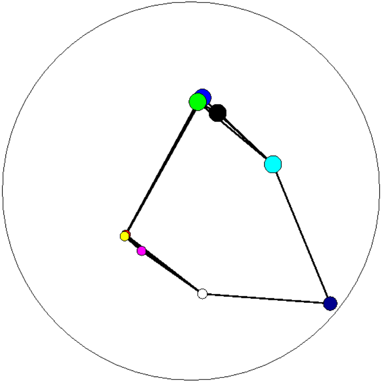


compared methods

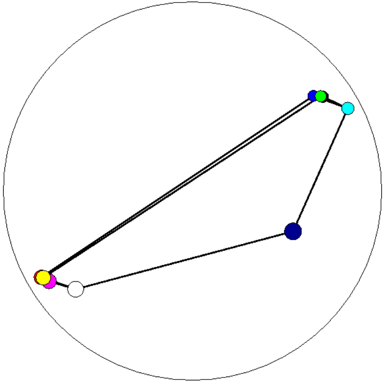


compared methods

Ncut (using Lagrange relaxation)

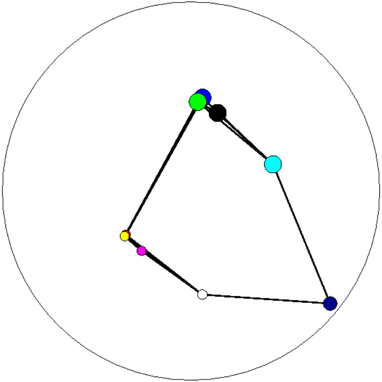


Modularity

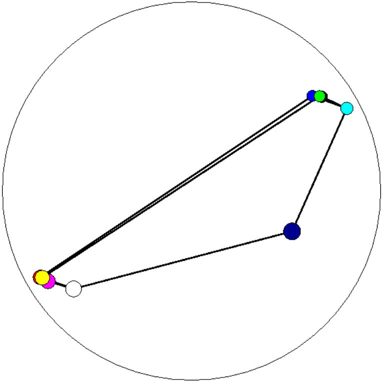


compared methods

Ncut (using Lagrange relaxation)

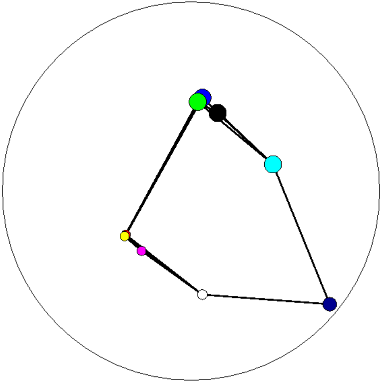


Modularity

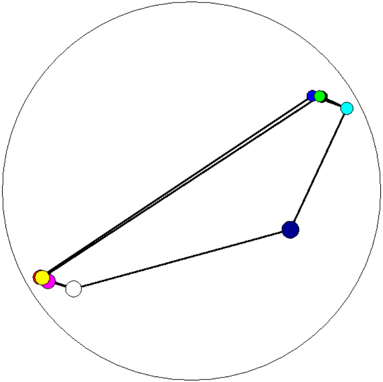


compared methods

Ncut (using Lagrange relaxation)

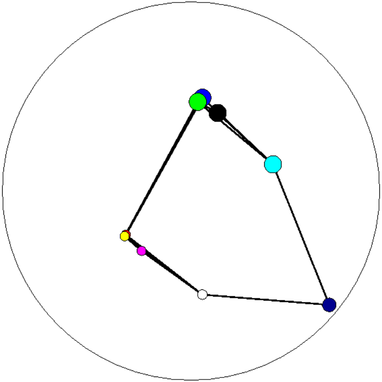


Modularity

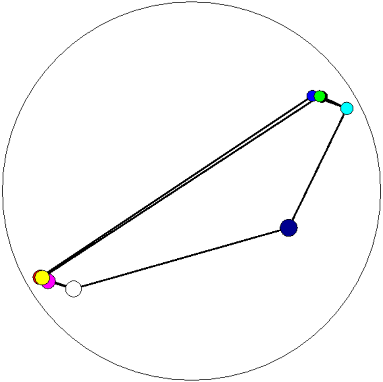


compared methods

Ncut (using Lagrange relaxation)

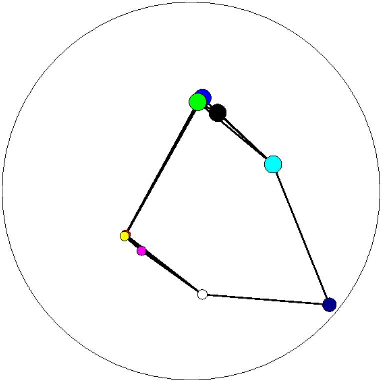


Modularity

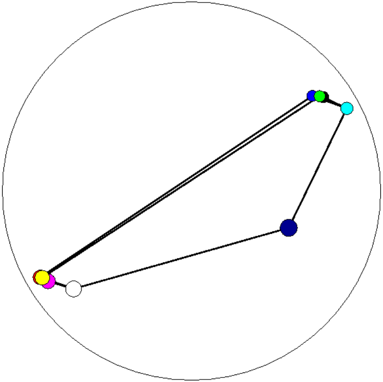


compared methods

Ncut (using Lagrange relaxation)

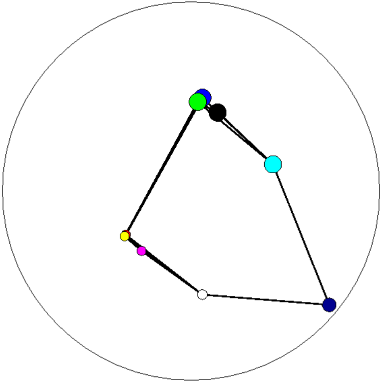


Modularity

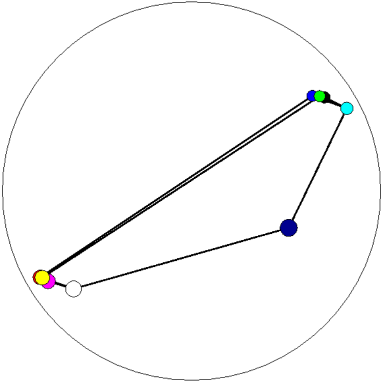


compared methods

Ncut (using Lagrange relaxation)

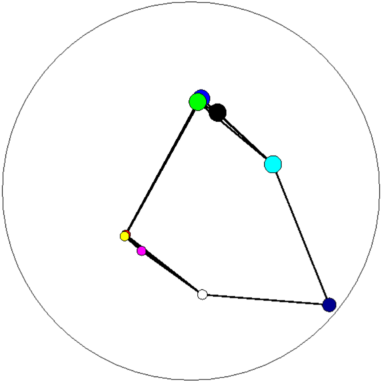


Modularity

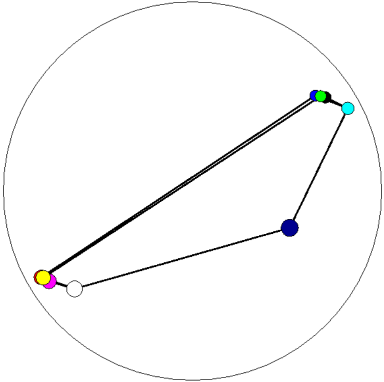


compared methods

Ncut (using Lagrange relaxation)

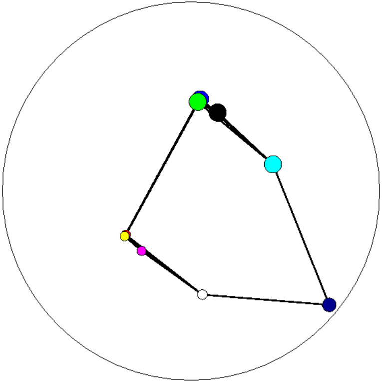


Modularity

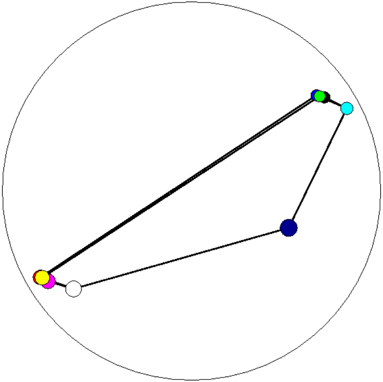


compared methods

Ncut (using Lagrange relaxation)

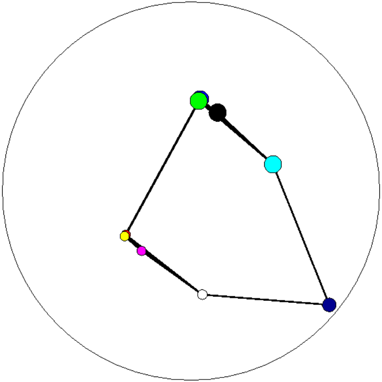


Modularity

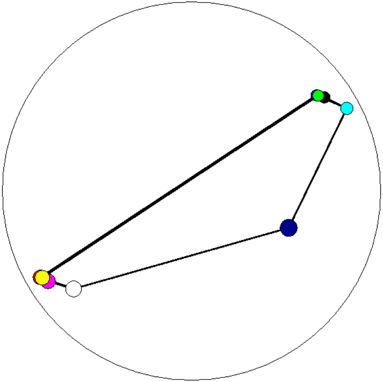


compared methods

Ncut (using Lagrange relaxation)



Modularity



→ following Goemans Williamson for Max-Cut, we cut with a random hyperplane begin

thinking further in terms of graphs

eigenvalues are simply great to split graphs !

thinking further in terms of graphs

eigenvalues are simply great to split graphs !

however social networks present often strongly connected components in terms of link density that should be detected to understand **communities**

thinking further in terms of graphs

eigenvalues are simply great to split graphs !

however social networks present often strongly connected components in terms of link density that should be detected to understand **communities**

how to detect them ?

orange™

thinking further in terms of graphs

eigenvalues are simply great to split graphs !

however social networks present often strongly connected components in terms of link density that should be detected to understand **communities**

how to detect them ?

the answer comes from matroid theory : let us analyse the strength of these graphs

what is the strength of a graph ?

Given a graph $G = (V, E)$, we compute

$$\sigma(G) = \min_{\mathcal{C} \text{ partition}} \frac{|\delta \mathcal{C}|}{p-1},$$

what is the strength of a graph ?

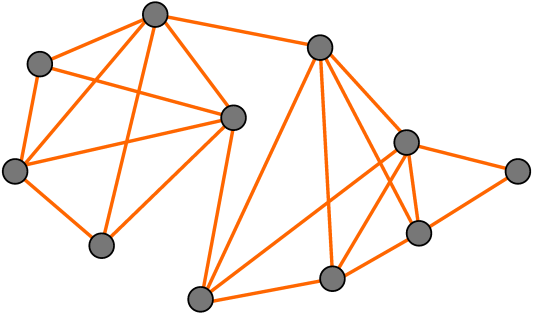
Given a graph $G = (V, E)$, we compute

$$\sigma(G) = \min_{\mathcal{C} \text{ partition}} \frac{|\delta \mathcal{C}|}{p - 1},$$

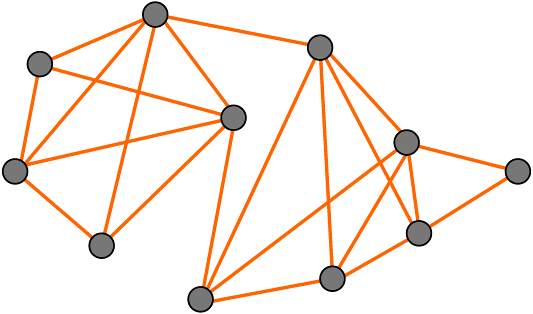
and the dual equivalent (the densest subgraph is terms of edges)

$$\gamma(G) = \max_{H \subseteq V,\ |H| \neq 1} \frac{|E(H)|}{|H| - 1},$$

Strength of graphs : intuition

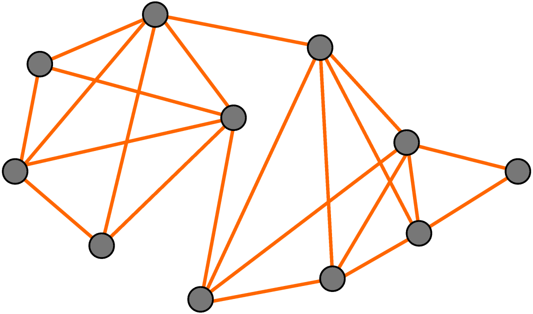


Strength of graphs : intuition



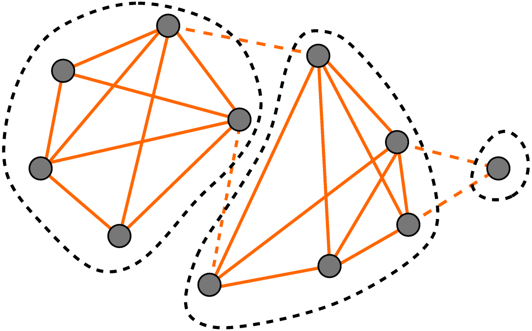
$\rightarrow$ minimize the ratio $\frac{\text{edges withdrawn}}{\text{created components}}$.

Strength of graphs : intuition



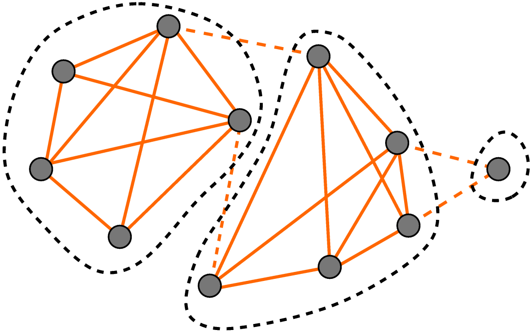
$\rightarrow$ minimize the ratio $\frac{\text{edges withdrawn}}{\text{created components}}$.

Strength of graphs : intuition



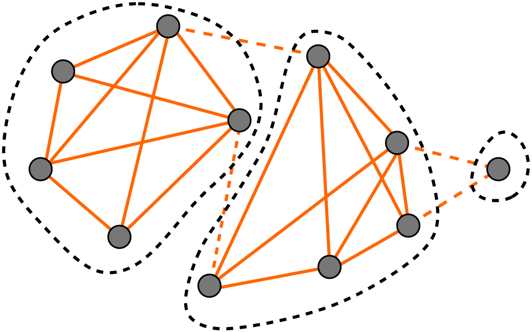
orange™

Strength of graphs : intuition



→ each sub-community that is not a singleton is then redivided and has provably a better strength.

Strength of graphs : intuition



→ each sub-community that is not a singleton is then redivided and has provably a better strength.

the Tutte Nash-Williams theorem (1961)

G contains k edge-disjoint spanning trees $\Leftrightarrow \sigma(G) \geq k$.

the Tutte Nash-Williams theorem (1961)

G contains k edge-disjoint spanning trees $\Leftrightarrow \sigma(G) \geq k$.

and for the dual :
$H \subseteq V$ that achieves $\gamma(G)$ contains $\lfloor \gamma(G) \rfloor$ spanning trees.

$$\text{Therefore } A.\mathbf{1}_H \geq \gamma(G)\mathbf{1}_H$$

the Tutte Nash-Williams theorem (1961)

G contains k edge-disjoint spanning trees $\Leftrightarrow \sigma(G) \geq k$.

and for the dual :
$H \subseteq V$ that achieves $\gamma(G)$ contains $\lfloor \gamma(G) \rfloor$ spanning trees.

Therefore $A.\mathbf{1}_H \geq \gamma(G)\mathbf{1}_H$

and so $\boxed{\Lambda_1 \geq \gamma(G)}$.

orange™

a word on the bibliography

Strength of graph is linked to *graph partitionning* and serves as the underground algorithm to approximate the *minimum cut* of a graph in almost linear time (Karger 2000).

Many algorithms use the maximum flow, which runs with best complexity $MF(n,m) = O(\min(\sqrt{m}, n^{2/3}) m \log(n^2/m+2))$ (Goldberg & Rao, 1998).

1984	Cunningham	$O(nm\ MF(n,n^2))$	Exact
1988	Gabow &	$O(\sqrt{\frac{m}{n}}(m+n\log n)\log\frac{m}{n})$	Integer
	Westermann	$O(nm\log\frac{m}{n})$	Integer
1991	Gusfield	$O(n^3 m)$	Exact
1991	Plotkin et ali	$O(m^2\sigma(G)\log(n)^2/n/\varepsilon^2)$	Within $1+\varepsilon$
1993	Trubin	$O(n\ MF(n,m))$	Exact
2008	Galtier	$O(m\log(n)^3/\varepsilon^2)$	Within $1+\varepsilon$
2011	Toko-Worou & Galtier	$O(m\gamma\log(n)^2/\varepsilon^2)$	γ within $1+\varepsilon$

A word on the linear approximation

The algorithm as basis takes a pushing flow scheme.

(0) Each edge $e \in E$ receives a very small weight $w(e) = \delta = O(n^{-3/\varepsilon})$,

A word on the linear approximation

The algorithm as basis takes a pushing flow scheme.

(0) Each edge $e \in E$ receives a very small weight $w(e) = \delta = O(n^{-3/\varepsilon})$,

(1) At each step, compute a minimum spanning tree T with respect to w,

A word on the linear approximation

The algorithm as basis takes a pushing flow scheme.

(0) Each edge $e \in E$ receives a very small weight $w(e) = \delta = O(n^{-3/\varepsilon})$,

(1) At each step, compute a minimum spanning tree T with respect to w,

(2) For each $e \in T$, update $w(e) := w(e) * (1 + \varepsilon)$,

A word on the linear approximation

The algorithm as basis takes a pushing flow scheme.

(0) Each edge $e \in E$ receives a very small weight $w(e) = \delta = O(n^{-3/\varepsilon})$,

(1) At each step, compute a minimum spanning tree T with respect to w,

(2) For each $e \in T$, update $w(e) := w(e) * (1 + \varepsilon)$,

(3) If $w(T) < 1$ go to (1),

A word on the linear approximation

The algorithm as basis takes a pushing flow scheme.

(0) Each edge $e \in E$ receives a very small weight $w(e) = \delta = O(n^{-3/\varepsilon})$,

(1) At each step, compute a minimum spanning tree T with respect to w,

(2) For each $e \in T$, update $w(e) := w(e) * (1+\varepsilon)$,

(3) If $w(T) < 1$ go to (1),

(4) Output $\sum_{e \in E} w(e)$.

orange™

A word on the linear approximation

The algorithm as basis takes a pushing flow scheme.

(0) Each edge $e \in E$ receives a very small weight $w(e) = \delta = O(n^{-3/\varepsilon})$,

(1) At each step, compute a minimum spanning tree T with respect to w,

(2) For each $e \in T$, update $w(e) := w(e) * (1 + \varepsilon)$,

(3) If $w(T) < 1$ go to (1),

(4) Output $\sum_{e \in E} w(e)$.

$\rightarrow$ this is an-$(1+\varepsilon)$ approximation
(Plotkin, Shmoys, Tardos 1991, Young 1995).

A word on the linear approximation

The algorithm as basis takes a pushing flow scheme.

(0) Each edge $e \in E$ receives a very small weight $w(e) = \delta = O(n^{-3/\varepsilon})$,

(1) At each step, compute a minimum spanning tree T with respect to w,

(2) For each $e \in T$, update $w(e) := w(e) * (1 + \varepsilon)$,

(3) If $w(T) < 1$ go to (1),

(4) Output $\sum_{e \in E} w(e)$.

$\rightarrow$ this is an-$(1+\varepsilon)$ approximation
(Plotkin, Shmoys, Tardos 1991, Young 1995).

Illustration of the algorithm

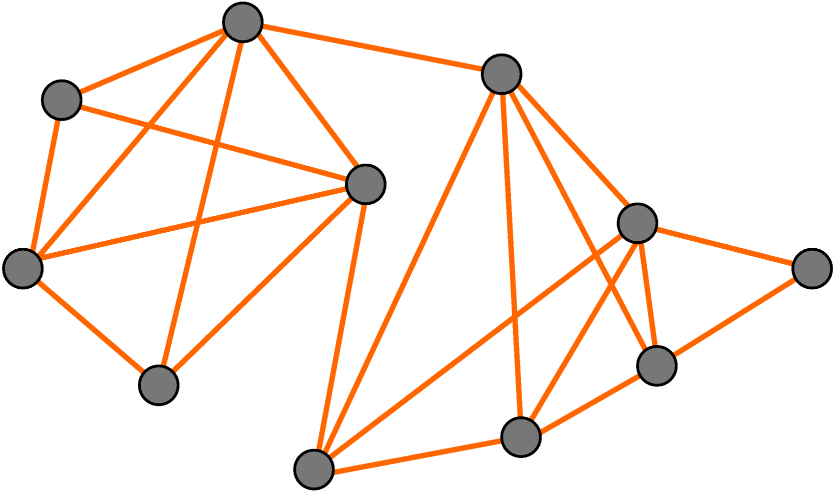


Illustration of the algorithm

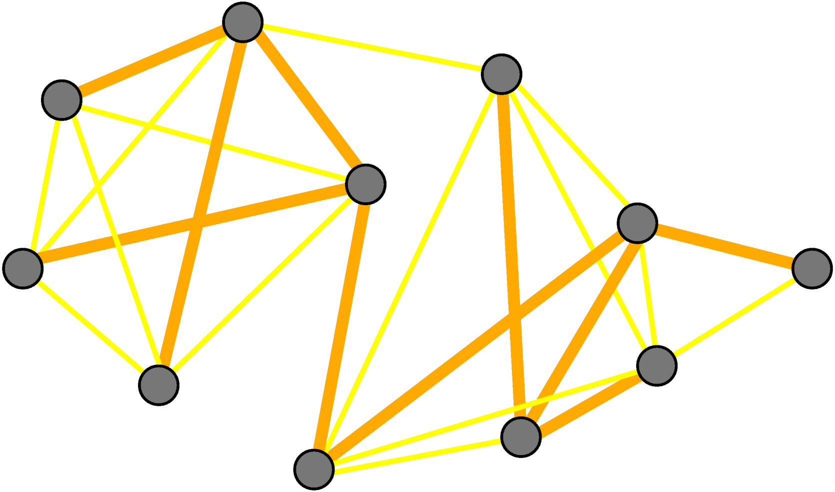
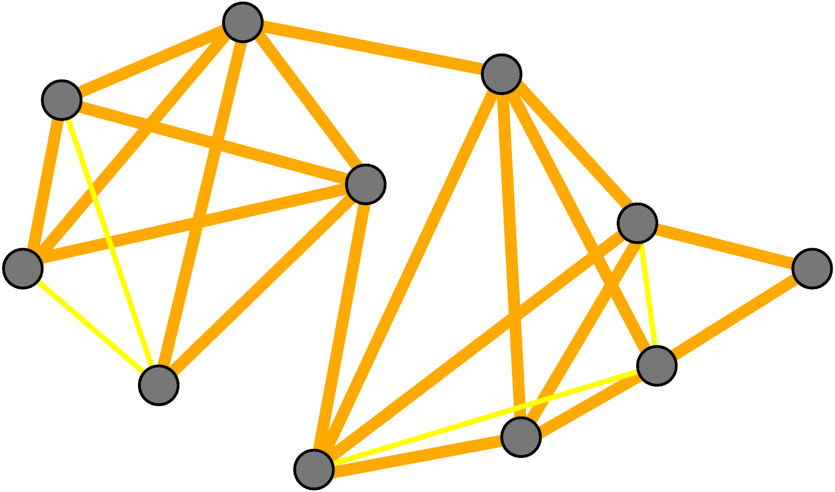


Illustration of the algorithm



orange™

Illustration of the algorithm

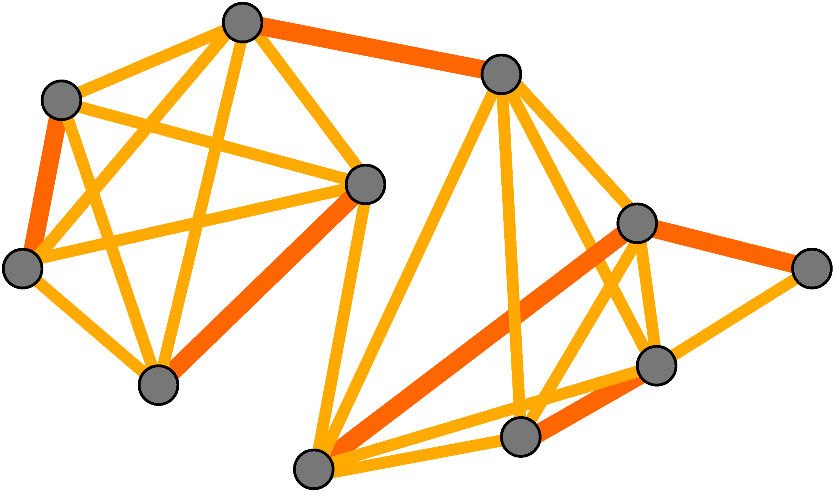


Illustration of the algorithm

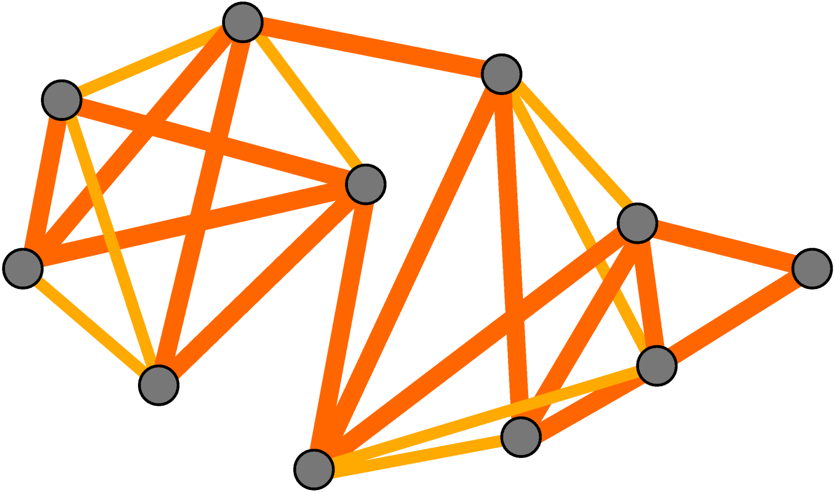


Illustration of the algorithm

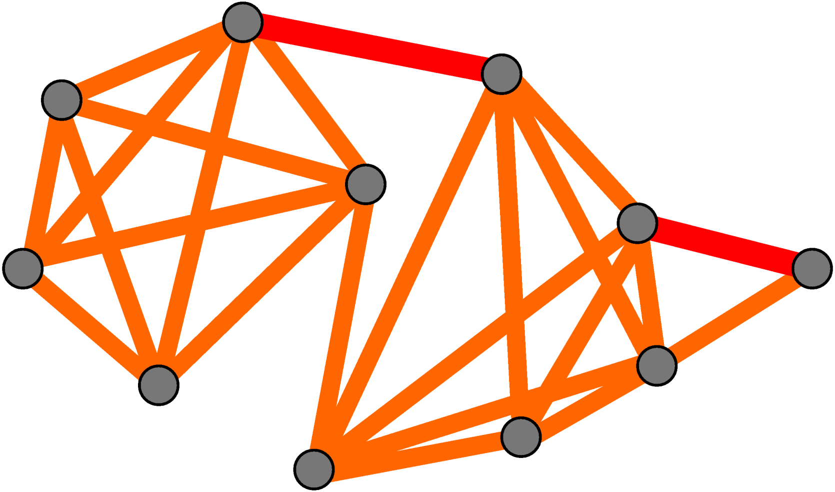


Illustration of the algorithm

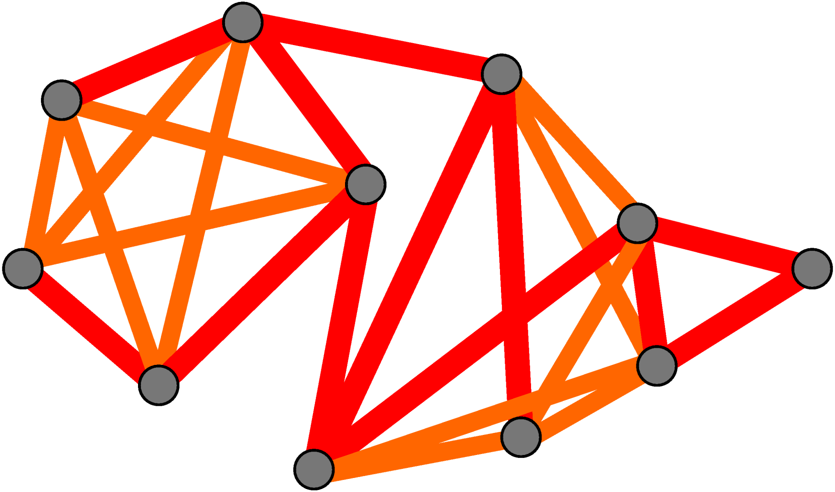


Illustration of the algorithm

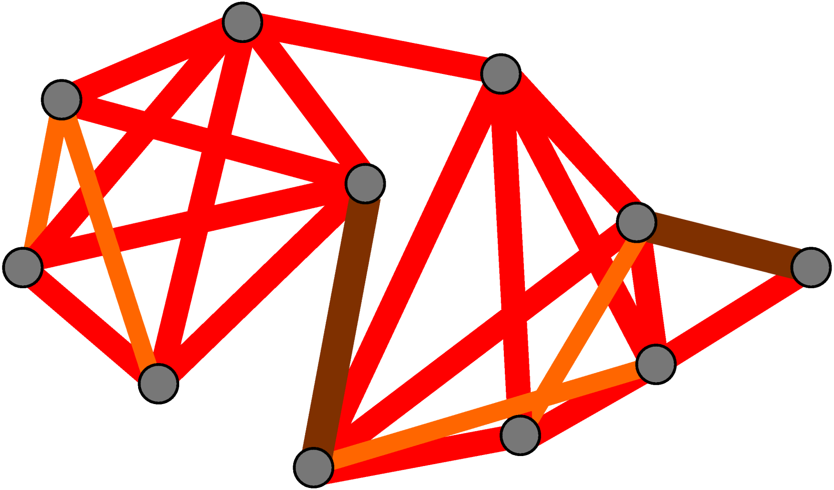


Illustration of the algorithm

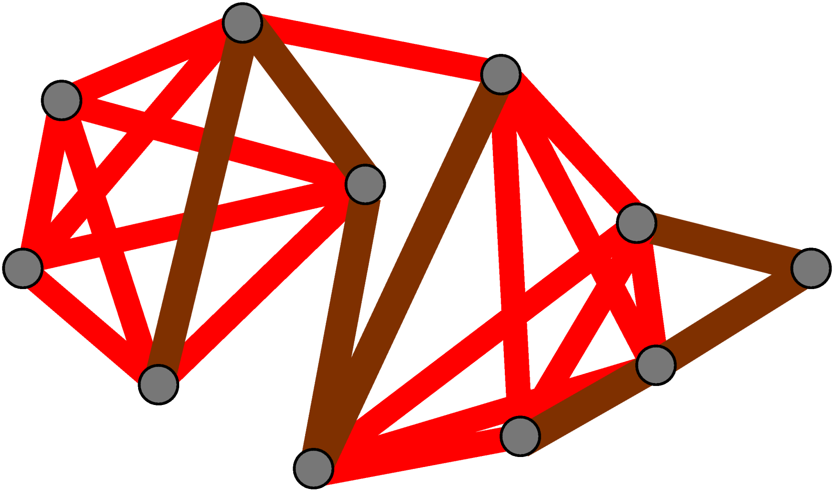


Illustration of the algorithm

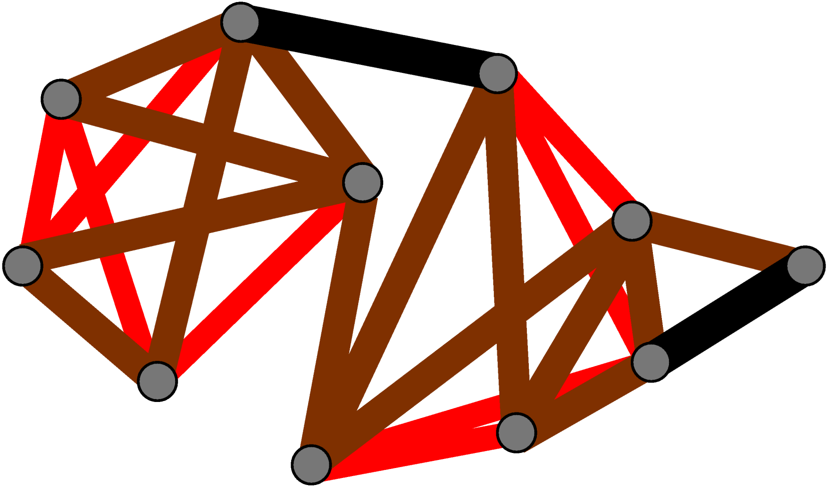
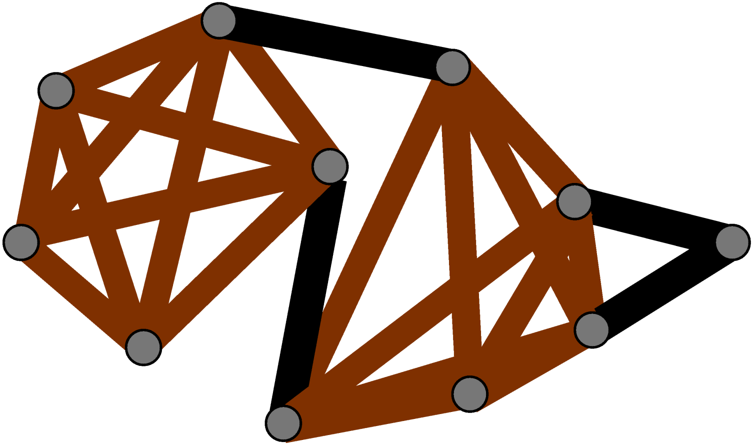
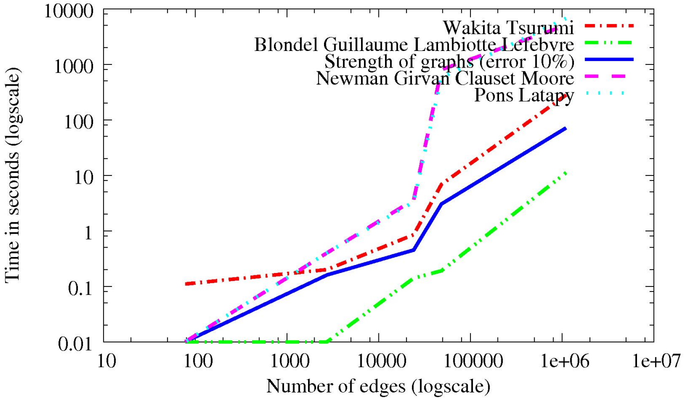


Illustration of the algorithm

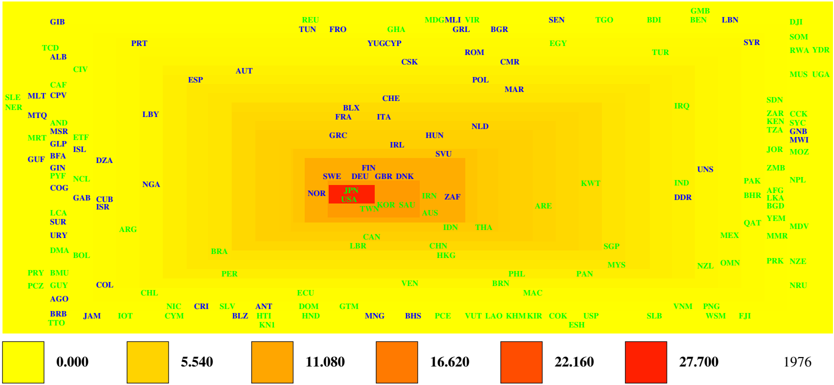


computational linearity of the approximation

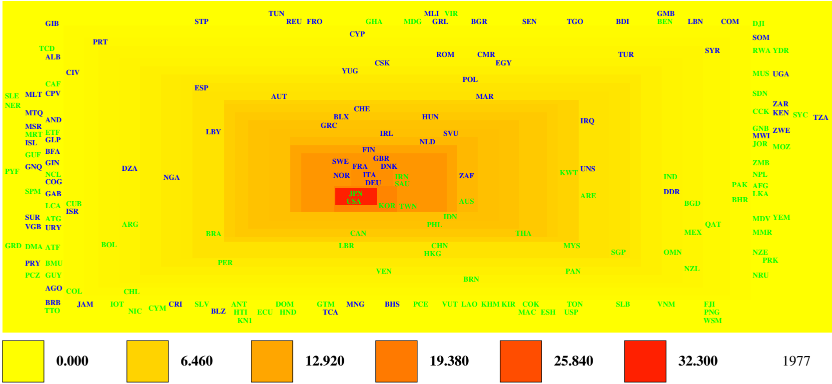
The algorithm is almost linear with the number of links between documents. Here compared with popular heuristics and datasets :



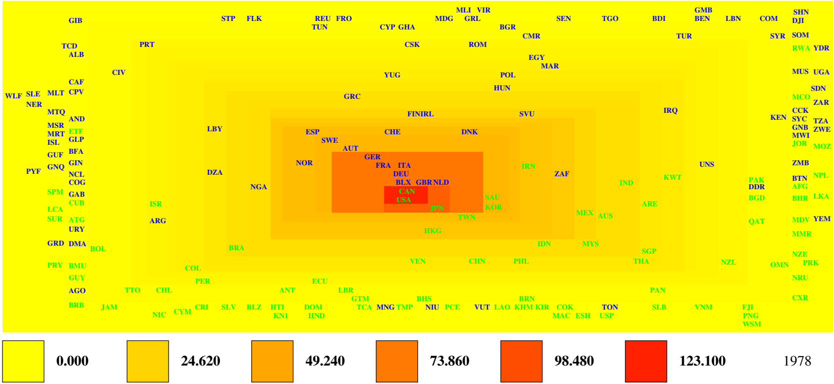
Back to bilateral exchanges



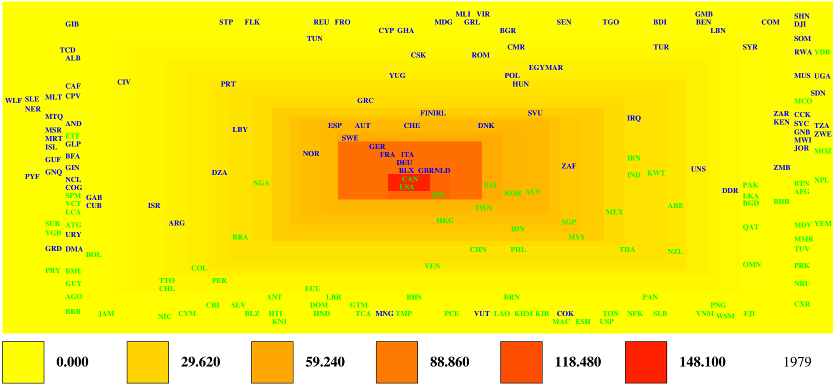
Back to bilateral exchanges



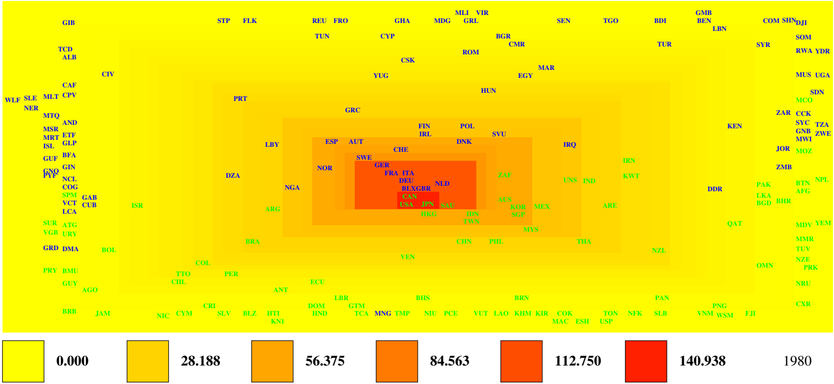
Back to bilateral exchanges



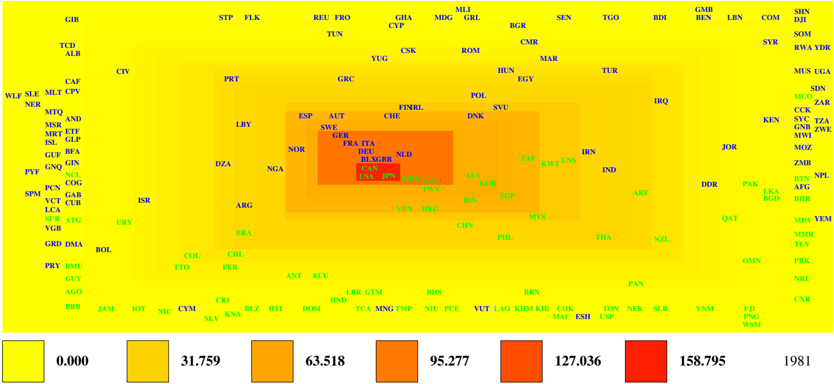
Back to bilateral exchanges



Back to bilateral exchanges

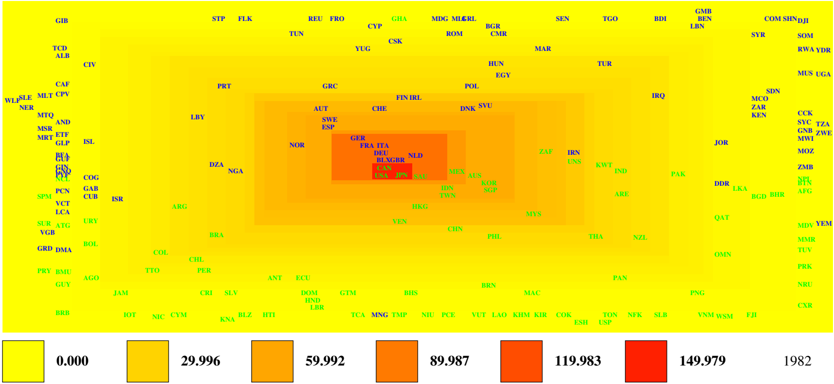


Back to bilateral exchanges

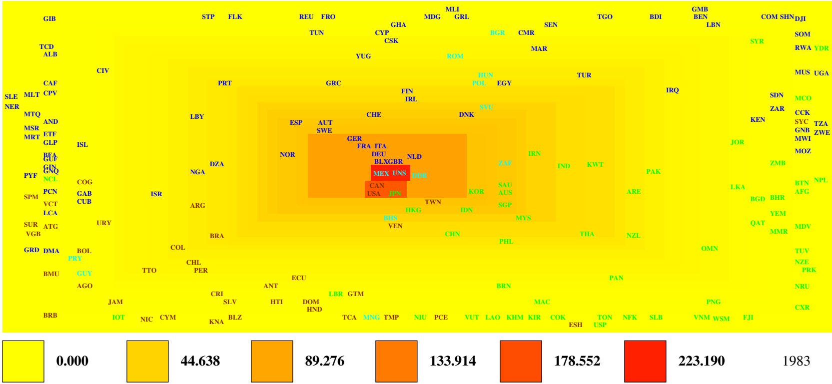


orange™

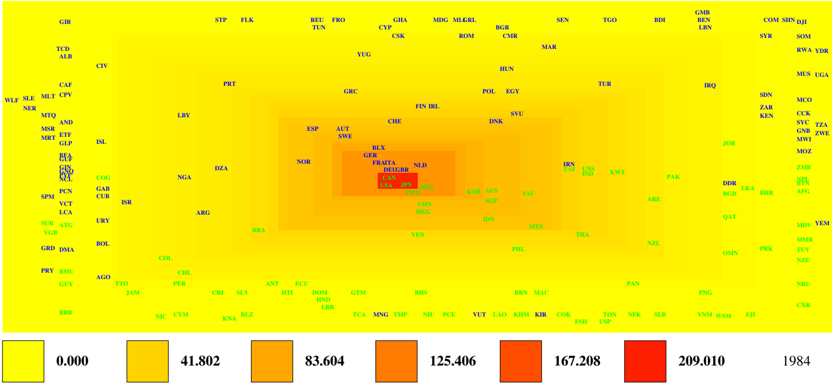
Back to bilateral exchanges



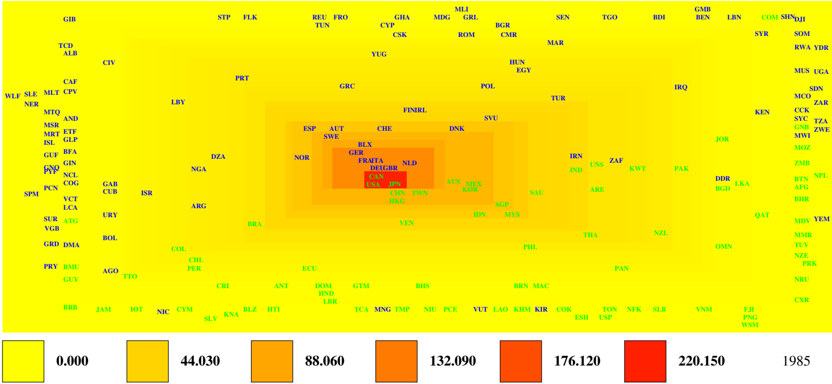
Back to bilateral exchanges



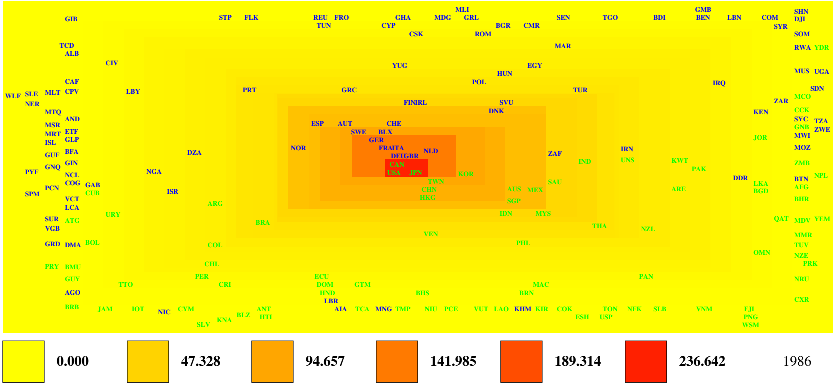
Back to bilateral exchanges



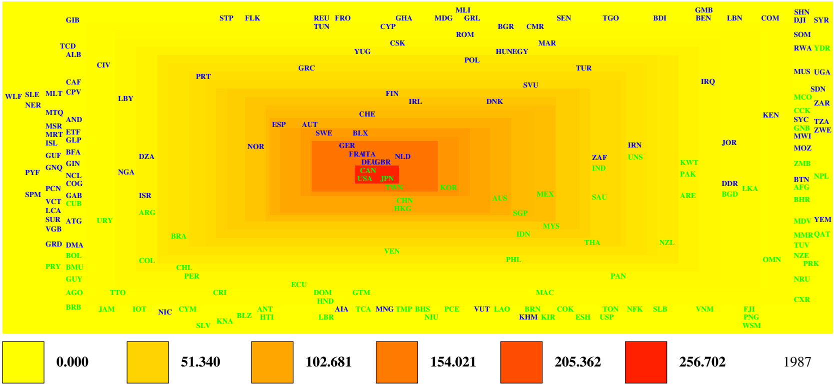
Back to bilateral exchanges



Back to bilateral exchanges

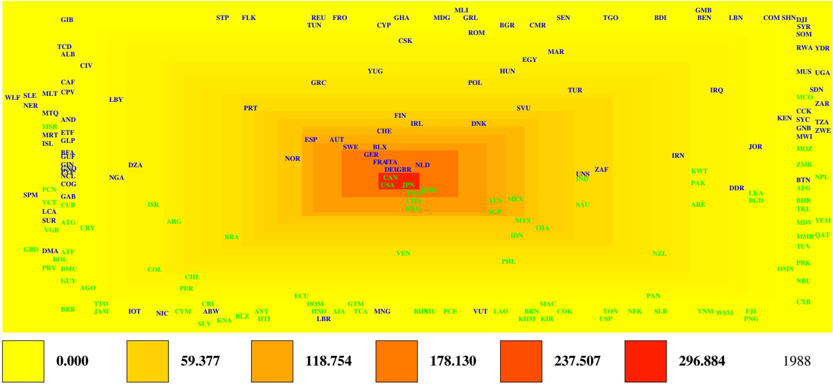


Back to bilateral exchanges

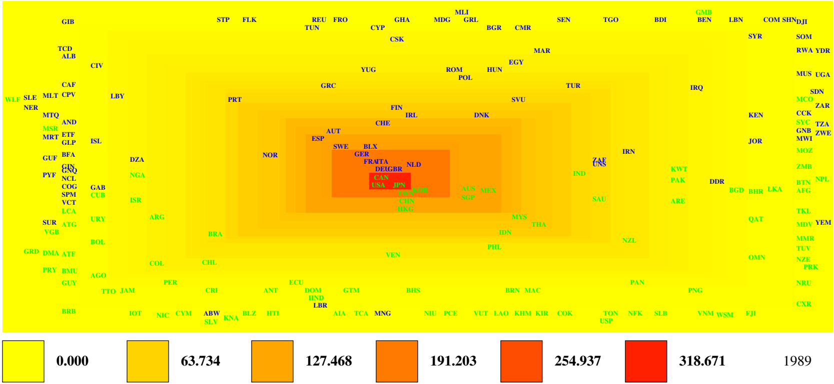


orange™

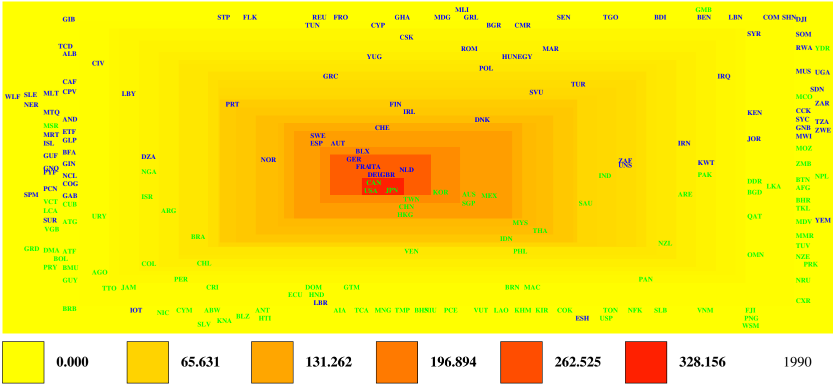
Back to bilateral exchanges



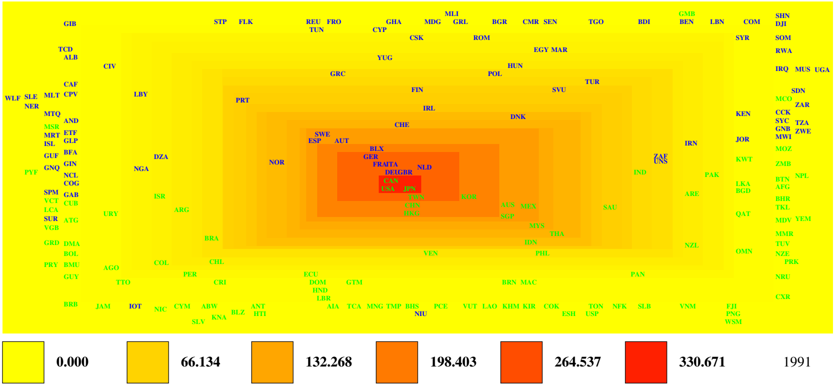
Back to bilateral exchanges



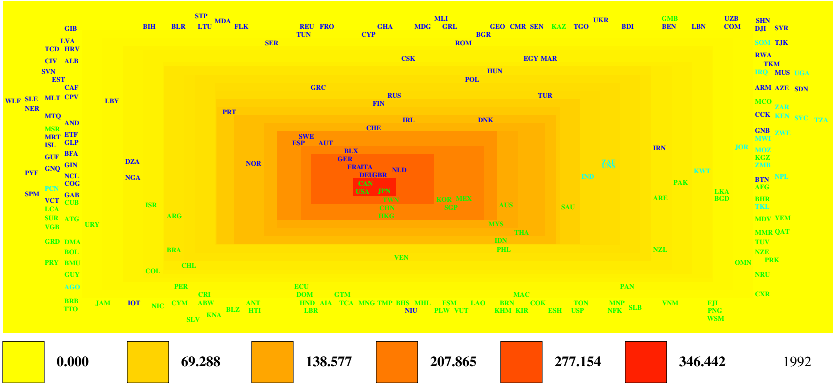
Back to bilateral exchanges



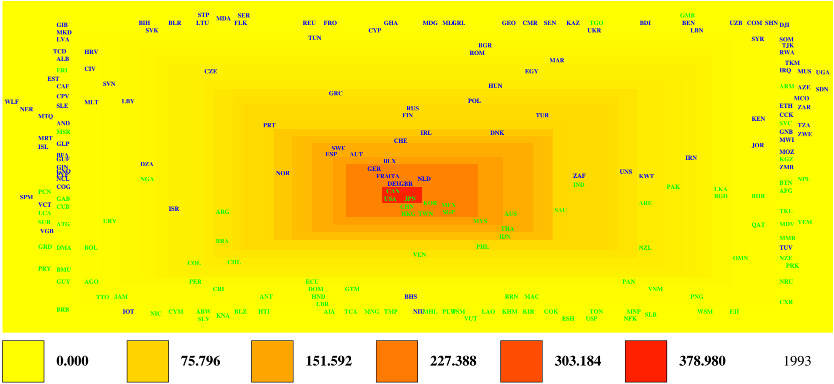
Back to bilateral exchanges



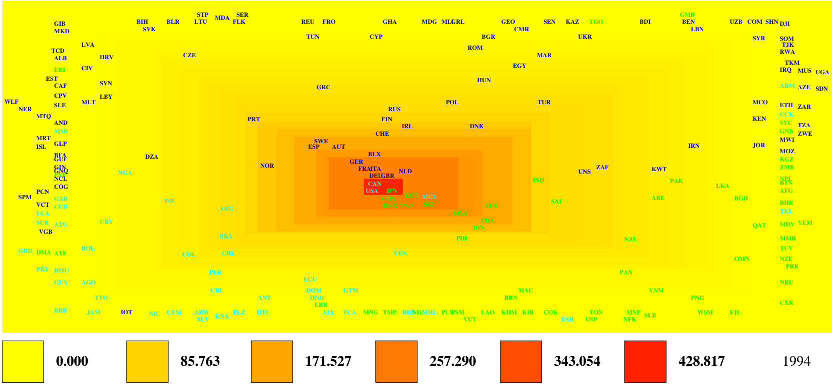
Back to bilateral exchanges



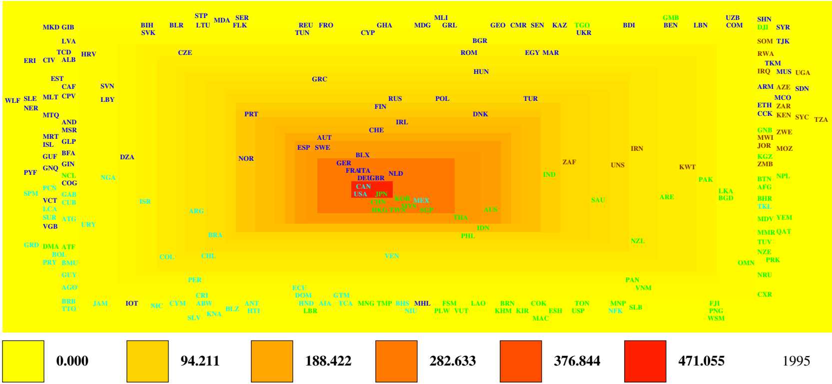
Back to bilateral exchanges



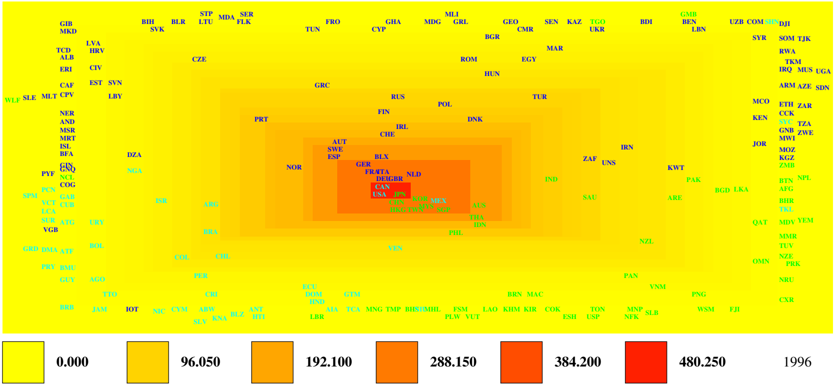
Back to bilateral exchanges



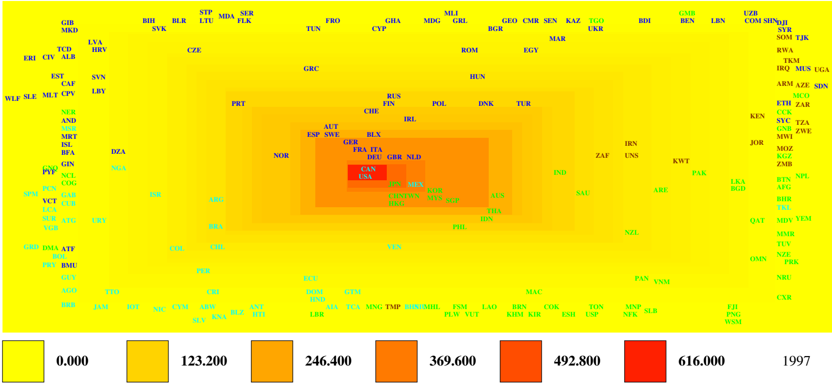
Back to bilateral exchanges



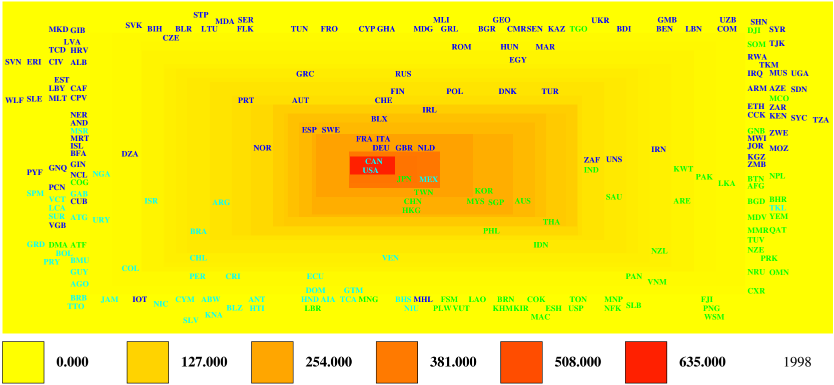
Back to bilateral exchanges



Back to bilateral exchanges



Back to bilateral exchanges



orange™

Back to bilateral exchanges

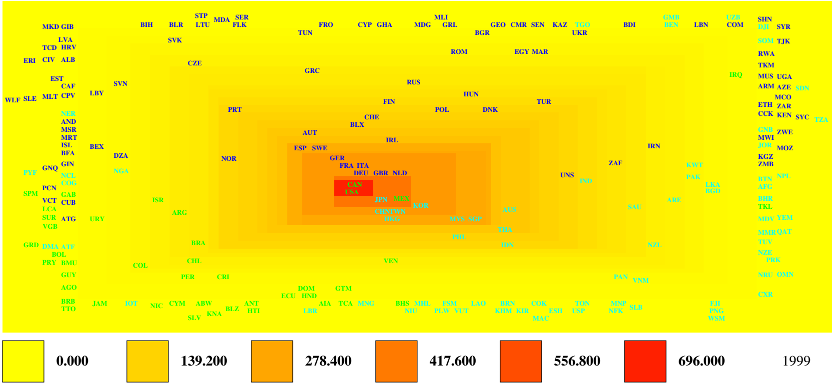
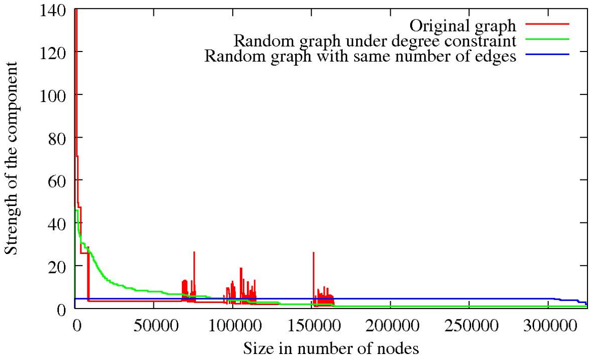


Diagram of the web



Conlusions

strength and modularity give nice analysis of graph communities

- first eigenvector of A is a relaxation of $\gamma(G)$
- second eigenvector of A is related (also with relaxations) to separation (*Cut*, *NCut* and Q)

and we have discussed of easy algorithms to compute them. . .

Questions

- what about directivity ?

orange™

Conlusions

strength and modularity give nice analysis of graph communities

- first eigenvector of A is a relaxation of $\gamma(G)$
- second eigenvector of A is related (also with relaxations) to separation (*Cut*, *NCut* and Q)

and we have discussed of easy algorithms to compute them...

Questions

- what about directivity ?
- how eigenvalues are further linked to (other) graph parameters ?

Conlusions

strength and modularity give nice analysis of graph communities

- first eigenvector of A is a relaxation of $\gamma(G)$
- second eigenvector of A is related (also with relaxations) to separation (Cut, $NCut$ and Q)

and we have discussed of easy algorithms to compute them. . .

Questions

- what about directivity ?
- how eigenvalues are further linked to (other) graph parameters ?

Thanks for four attention ! ! !

(and may the strength be with us)

Brute analysis of the complexity

Each edge cannot be updated more that $\frac{\log(\delta)}{\log(1+\varepsilon)} = O(\frac{\log(n)}{\varepsilon^2})$,
Each step updates $n-1$ edges and runs in $O(m\log(n))$,
$\rightarrow$ the computation takes less than $O(\frac{m^2\log(n)^2}{n\varepsilon^2})$.

orange™

Brute analysis of the complexity

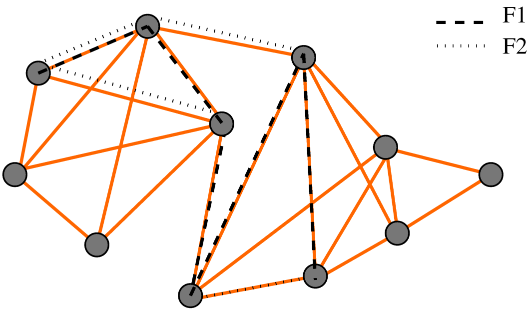
Each edge cannot be updated more that $\frac{\log(\delta)}{\log(1+\varepsilon)} = O(\frac{\log(n)}{\varepsilon^2})$,
Each step updates $n-1$ edges and runs in $O(m\log(n))$,
$\rightarrow$ the computation takes less than $O(\frac{m^2\log(n)^2}{n\varepsilon^2})$.
how can we gain the factor m/n ? ? ?

Brute analysis of the complexity

Each edge cannot be updated more that $\frac{\log(\delta)}{\log(1+\varepsilon)} = O(\frac{\log(n)}{\varepsilon^2})$,
Each step updates $n-1$ edges and runs in $O(m\log(n))$,
$\rightarrow$ the computation takes less than $O(\frac{m^2\log(n)^2}{n\varepsilon^2})$.
how can we gain the factor m/n ? ? ?

order on forests

A forest F_1 is more connecting than a forest F_2 ($F_1 \succeq F_2$) if the endpoints of any path of F_2 are connected in F_1.



augment and connecting order

Let $e \in E$. We say that e is independent of forest F if there is no path in F between endpoints of e. Otherwise it is dependent.

augment and connecting order

Let $e \in E$. We say that e is independent of forest F if there is no path in F between endpoints of e. Otherwise it is dependent.

Augmenting F by an independent edge e to F : $F := F \cup \{e\}$.

augment and connecting order

Let $e \in E$. We say that e is independent of forest F if there is no path in F between endpoints of e. Otherwise it is dependent.

Augmenting F by an independent edge e to F : $F := F \cup \{e\}$.

Remark : Suppose $F_1 \succeq F_2$ and e is independent of F_1, then e is independent of F_2.

edge addition on ordered forests

idea : order the forests to add edges

$F_1 \succeq F_2 \succeq \cdots \succeq F_p$

take $e \in E$.

augment the first F_i such that e is independent to F_i.

edge addition on ordered forests

idea : order the forests to add edges

$F_1 \succeq F_2 \succeq \cdots \succeq F_p$

take $e \in E$.

augment the first F_i such that e is independent to F_i.

$\rightarrow$ a tree will be built in $O(n \log(n) \log(p))$ in average.

edge addition on ordered forests

idea : order the forests to add edges

$F_1 \succeq F_2 \succeq \cdots \succeq F_p$

take $e \in E$.

augment the first F_i such that e is independent to F_i.

$\rightarrow$ a tree will be built in $O(n \log(n) \log(p))$ in average.

$\rightarrow$ this works also with weighted edges in increasing order.

edge addition on ordered forests

idea : order the forests to add edges

$F_1 \succeq F_2 \succeq \cdots \succeq F_p$

take $e \in E$.

augment the first F_i such that e is independent to F_i.

$\rightarrow$ a tree will be built in $O(n\log(n)\log(p))$ in average.

$\rightarrow$ this works also with weighted edges in increasing order.

this gives an $O(m\log(n)^3/\varepsilon^2)$ algorithm.

edge addition on ordered forests

idea : order the forests to add edges

$F_1 \succeq F_2 \succeq \cdots \succeq F_p$

take $e \in E$.

augment the first F_i such that e is independent to F_i.

$\rightarrow$ a tree will be built in $O(n \log(n) \log(p))$ in average.

$\rightarrow$ this works also with weighted edges in increasing order.

this gives an $O(m \log(n)^3/\varepsilon^2)$ algorithm.